MASTER EN SCIENCES INFORMATIQUES

Rétro-ingénierie de la variabilité au sein de lignes de produits logiciels web: une "systematic mapping study"

Patiny, Mathieu

Award date:
2015

Awarding institution:
Universite de Namur

UNIVERSITÉ DE NAMUR
Faculté d'informatique
Année académique 2014-2015

Rétro-ingénierie de la variabilité au sein de lignes de produits logiciels web : une « systematic mapping study »

Mathieu PATINY

Promoteur ———————— : (Signature pour approbation du dépôt - REE art. 40)
Professeur Patrick HEYMANS

Co-promoteurs : Monsieur Xavier DEVROEY et Docteur Gilles PERROUIN

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Résumé

Dans la société actuelle, la réutilisation d'artéfacts (parties de code source ou d'analyse) dans le développement logiciel est de plus en plus présente. Les systèmes développés sur base des mêmes artéfacts sont appelés « produits » et forment, ensemble, une « ligne de produits logiciels ». Cette nouvelle manière de travailler est plus complexe que l'approche suivie dans un développement traditionnel. C'est dans cette direction que sont menées certaines recherches, comme la vérification automatisée de ces systèmes. Ce travail de fin d'études recense, entre autres, des techniques de rétro-ingénierie d'expression de la variabilité (ciblant les différences et similarités entre les produits) au sein de systèmes web. La méthode « systematic mapping study » a été appliquée pour réaliser ce travail. Cette méthode est une démarche protocolaire permettant de classifier des publications afin de répondre à des questions de recherches. Les premiers résultats ont démontré le manque de littérature orientée systèmes web. Ce constat a mené à l'extension des recherches aux systèmes en général. Sur base de plus de 1500 publications récupérées de manière automatique, 72 ont, au final, été exploitées et ont mis en évidence différentes techniques d'extraction de la variabilité au sein de ces systèmes. Il a été conclu que les techniques existantes permettent depuis 8 types de sources d'entrée, de retourner 11 types d'artéfacts de sortie différents. Le plus grand nombre de ces techniques se basent sur le code source (sa structure, des annotations ou commentaires particuliers, etc.) des produits en voulant obtenir des modèles (spécialement de type feature model). Chaque technique a été aussi reliée aux types de recherches effectuées (proposition, expérimentation, validation et évaluation).
Une analyse plus poussée a aussi permis de lister :

- les mécanismes d'expression de variabilité (dont les plus connus sont les cross-tree constraints, les annotations spécifiques du code source, les modèles particuliers comme les implications (hyper)graphs, etc.).
- les artéfacts intermédiaires des techniques (comme l'analyse FCA et des algorithmes très souvent empiriques mais aussi différents types de formules).
- des outils spécifiques (tels que « FAMA tool suite », « FAMILIAR », etc.).

Ce mémoire est destiné à tout praticien désirant étudier une technique particulière de rétro-ingénierie ou tout chercheur souhaitant explorer de nouvelles techniques.

Mots clés : *Ligne de produits logiciels, rétro-ingénierie, variabilité logicielle, commonality, système web, feature model, feature transition system, systematic mapping study, artéfacts de rétro-ingénierie.*

Avant-propos

Ce mémoire est le résultat d'une tâche de longue haleine et d'études approfondies sur le sujet abordé. Au terme de ce mémoire, j'aimerais adresser mes sincères remerciements à toutes les personnes ayant d'une façon ou d'une autre contribué à l'élaboration de ce travail :

- En premier lieu, M X. Devroey et G. Perrouin avec qui et pendant de multiples heures, le sujet a pris forme et évolué tout au long de cette année.
  Avec un remerciement tout particulier à M Devroey pour toutes les relectures, commentaires et le suivi très serré de chaque partie décrite dans ce présent mémoire.

- M M. Cordy pour avoir proposé l'idée originale d'orienter ce travail vers la rétro-ingénierie.

- M A. Cleve pour avoir renseigné de la litterature permettant d'effectuer une partie de l'état de l'art de ce mémoire.

- M P. Heymans pour avoir proposé et permis le choix de ce mémoire.

- L'ensemble de mes professeurs en informatique qui, par leur transfert de savoirs et compétences, m'ont donné toutes les bases nécessaires pour mener à bien ce projet.

J'aimerais aussi dédier un remercîment tout particulier aux membres de ma famille qui ont su, en toutes circonstances, me soutenir tout au long de cette année de travail (moralement ou encore par le biais de délicieux repas me permettant de reprendre des forces pour travailler sans relâche).
Avec un toute particulière attention envers Alain, mon papa pour ces intensives relectures ainsi que Florentine et Antoine, ma sœur et mon beau-frère pour cette même raison.

A tous ces intervenants, je tiens donc à présenter toute ma gratitude et mes remerciements.

Table des matières

Tables des illustrations

Table des Figures

Table des Tableaux

Mots clés

- AST : Abstract Syntax Tree
- CMS : Content Management System
- CNF : Forme normale conjonctive
- DOM : Document Object Model
- FCA : Formal Concept Analysis
- FD(s) : Feature Diagram(s)
- FM(s) : Feature Model(s)
- FTS(s) : Feature Transition System(s)
- LPL : Ligne de Produits Logiciels
- LSI : Latent Semantic Indexing
- QR : Questions de Recherche
- RCA : Relational Concept Analysis
- RIA : Rich Internet Application
- RIV : Rétro-Ingénierie de Variabilité
- SHV : Système à Haute Variabilité
- TS(s) : Transition System(s)
- TVL : Text-based Variability Language
- UML : Unified Modeling Language
- VDE : Variability Data Extraction
- XML : Extensible Markup Language

Partie 1 : introduction

1 Contexte global de recherche

Ce mémoire s'inscrit dans le contexte de recherche très large qu'est la vérification automatisée de systèmes sous forme de lignes de produits logiciels. Dans cette optique, certains chercheurs, comme à l'Université de Namur, entament des travaux permettant la production de modèles représentant le comportement exact d'un de ces systèmes. Dans ce but, une piste plausible et cohérente est d'effectuer une rétro-ingénierie depuis ces derniers. Ce mémoire a comme motivation première de venir ajouter une pierre à cet édifice en explorant et regroupant les pistes déjà proposées concernant la rétro-ingénierie de la variabilité de ces systèmes et plus précisément pour les systèmes web. La rétro-ingénierie de la variabilité étant dans ce cas, un premier pas vers la rétro-ingénierie de modèles comportementaux.

2 Contexte du mémoire

Le terme ligne de produits logiciels provient directement de la métaphore des lignes de produits industriels. Tout comme elles, ce paradigme permet de prôner la production de produits sur base d'artéfacts existants dans le but de voir le coût de chaque produit diminuer.

Le développement sous le paradigme des lignes de produits logiciels apporte beaucoup d'avantages. Mis à part l'avantage financier premier exprimé dans ce mémoire, le second grand avantage est de gérer entièrement la réutilisation d'artéfacts au sein d'un système. Ce principe entre dans le cadre de notre société économique tendant à développer des systèmes sur base d'artéfacts préconstruits, que ce soit au niveau de la réutilisabilité de codes source (librairies, partage d'objets ... mais aussi un simple copier/coller), mais également de tout élément provenant de son analyse (pattern(s), partie(s) de schémas de bases de données ...). Ce paradigme vient encadrer et répondre aux nouveaux problèmes soulevés par cette manière particulière de procéder.

Lors du développement respectant ce paradigme, la gestion de la variabilité est un point essentiel et critique. C'est en effet cette variabilité qui exprime l'ensemble des différences et points communs qu'ont tous les produits de la ligne de produits logiciels. Ce point devient d'autant plus important que les applications de type web à haute variabilité (ex. : systèmes de type CMS totalement configurables via le système des plugins) se multiplient à grande vitesse.

Dans tout cycle de vie d'un système, la phase de maintenance (corrections et améliorations) est une étape critique et très coûteuse en temps et argent. Cette phase l'est d'autant plus dans les systèmes sous forme de lignes de produits logiciels que leurs analyses sont complexes et peuvent pourtant être très légères (ex. : startup dynamique de production de prototype). Une solution est de recréer une partie de cette analyse sur base d'éléments disponibles. C'est dans cette optique que ce mémoire tente d'apporter des solutions.

Cette recherche de rétro-ingénierie de la variabilité étant très récente, il va de soi que le niveau de maturité est encore faible et que les approches proposées sont multiples et variées. C'est dans cette optique qu'une recherche (la plus complète possible) des techniques existantes est effectuée dans ce mémoire. C'est sous la forme de techniques très protocolaires que sont les « systemtics mapping studies » (recherches visant à répondre à un besoin de classification de la littérature) que cette recherche est menée. L'analyse de ces résultats permettra de mettre en évidence les différentes techniques explorées, mais aussi de les replacer dans leur contexte : sont-elles juste proposées dans une publication ? Si oui, de quand datent-elles ? Ont-elles déjà été implémentées et testées ? Si oui, dans quelles publications ? Sur quel(s) outil(s) particulier(s) se basent-elles ? Ont-elles été évaluées et validées par d'autres chercheurs ? Etc.

Ce mémoire tente de répondre au mieux à ces multiples questions et d'interpréter les résultats pour mettre en évidence l'état d'avancement des recherches déjà menées, et proposer des directions viables pour les futures recherches.

Ce mémoire est constitué de 3 grandes sections :

- le contexte dans lequel prend naissance ce mémoire est complètement fixé et décrit dans les détails.

- la seconde section reprend l'entièreté de l'état de l'art du paradigme des lignes de produits logiciels et du point sensible qu'est la variabilité. Un modèle particulier et très répandu jusqu'à en être incontournable (feature model) permettant d'exprimer cette dernière est mis en avant. Ainsi, sa syntaxe et sémantique sont expliquées en profondeur. Une multitude d'opérations applicables sont aussi présentées. Un processus permettant la correction de ce type de modèle par rapport à un système déjà implémenté est aussi exposé et mis en relation avec la philosophie des techniques de rétro-ingénierie.
  C'est ensuite un tour d'horizon de la rétro-ingénierie depuis ses origines et ses objectifs appliqués à ce domaine qui est proposé. Cette vision permet, entre autres, de déterminer en quoi ces techniques répondent aux besoins actuels de la société économique.
  Une idée originale et basée sur les modèles nommés « feature transition system » et non encore expérimentée pour cette application est ensuite mise en avant.

- la troisième section propose l'état des lieux de ce domaine. Elle reprend de manière parallèle, l'explication détaillée d'un protocole de « systematic mapping study » et son application dans un cas réel. Ainsi, étape par étape, les recherches se construisent et forment un ensemble de résultats interprétés en fin de section. C'est par le biais de ces interprétations que des réponses ou éléments de réponses aux questions de départ pourront être proposés.

Partie 2 : au cœur du mémoire

Chapitre I. Un contexte particulier

1 Le contexte particulier qu'est de la rétro-ingénierie des lignes de produits logiciels web

Dans le cadre de ce mémoire, les systèmes web ubiquitaires dans notre société de l'information seront au centre du sujet. En effet, certaines entreprises exercent leurs activités uniquement grâce aux revenus financiers que génère leur système web. Pour citer un exemple bien connu qu'est le site de vente en ligne Amazon, ce sont 1 837 000 visiteurs (fédération e-commerce et vente à distance 2014) uniques en moyenne par jour avec un record atteignant 1 million de colis expédiés depuis la France (le 8 décembre 2014). Avec un tel volume d'échanges, la capacité de pouvoir comprendre et analyser ces systèmes, afin de faciliter leur évolution et de détecter des problèmes éventuels, est critique pour ces entreprises. De fait, on peut ranger ces systèmes dans la catégorie « business-critical » (Sommerville 2009), c'est-à-dire, dont les défaillances peuvent engendrer des pertes considérables pour les entreprises les fournissant, et parfois conduisant à leur faillite.

Une approche possible et envisageable serait de considérer ces systèmes comme des systèmes « safety-critical » (ces types de systèmes étant considérés comme critiques au plus haut niveau, ils sont toujours totalement analysés afin de ne laisser aucun bug possible) et de procéder à leur conception de manière rigoureuse en appliquant des méthodes formelles d'analyse et de génération à partir de leurs spécifications.
Cette approche n'est, hélas, pourtant pas réaliste, à cause du fait que la nature de ces applications est, par essence, évolutive (dans un certain nombre de cas, plusieurs évolutions par semaine peuvent avoir lieu) et de la culture des développeurs. En effet, ces applications sont souvent le fruit de startups ou le résultat d'un petit groupe de développeurs prototypant des idées dans le cadre de processus de type « Agile ». Dès lors, il n'est pas vraiment envisageable de comprendre et valider les systèmes « à priori ». Par contre, il est plus judicieux d'étudier ces systèmes « à postériori ». Une analyse après construction de ces systèmes permettrait d'en extraire des éléments de leurs spécifications inexistantes. Pour ce, il faut dès lors se tourner vers des techniques dites de « rétro-ingénieries ».

La maintenance et l'évolution des applications deviennent de plus en plus une préoccupation majeure des entreprises due aux coûts importants et à l'ampleur du travail qu'elles engendrent. En effet, la maintenance d'un logiciel constitue une partie importante du coût total de son cycle de vie. Ce coût représente entre 50 % et 80 % du coût total du cycle de vie (HACHICHI 2011). A l'heure actuelle, les entreprises se voient confrontées à des logiciels dont 80 % ne sont ni structurés, ni corrigés et encore trop peu documentés. Seul un processus de maintenance pourrait combler ces problèmes. La rétro-ingénierie telle que définie par la suite apporte aussi une grande avancée dans ce processus de maintenance.
Que ce soit pour une application web ou un autre type d'application, la maintenance reste un processus obligatoire. Ce processus est d'autant plus important dans les systèmes web car ils sont souvent amenés à être développés par des dizaines, voire des centaines de personnes (de tous niveaux), dans les cas de systèmes composés de plugins/modules venant de communautés plus ou moins larges (voir ci-après) et pouvant être utilisés par de jeunes apprentis.

De plus en plus, les entreprises proposent différentes versions (versions en terme d'historique d'évolution d'un logiciel ou encore en terme de fonctionnalités) de leurs applications. Dans cette optique, un concept mettant en avant la réutilisation d'artéfacts d'une application à une autre afin de ne pas réinventer la roue pour chaque application est prônée. Ce paradigme se nomme « lignes de produits logiciels » et s'inspire en fait, d'un paradigme bien connu dans le monde matériel (industriel) qu'est celui des chaînes de production industrielle (Ziadi 2012). C'est un paradigme fort recherché par les entreprises dans le sens où de grandes économies financières peuvent être faites. C'est vers cette optique très attrayante que ce mémoire s'oriente.

Les approches via rétro-ingénieries peuvent être multiples car ce terme englobe une multitude de buts possibles dans des domaines variés (rétro-ingénierie d'objet matériel, humaine, électronique, logicielle ...). Ce mémoire a une orientation informatique et logicielle. Dans ce mémoire, il a été choisi de s'orienter vers des techniques de rétro-ingénierie permettant de comprendre la « variabilité ». Néanmoins, il est possible que des techniques intermédiaires de rétro-ingénierie autres que logicielles (comme la rétro-ingénierie sur les développeurs afin de comprendre comment certaines parties ont été développées) soient proposées. Ce concept de variabilité est d'une grande importance dans le cadre de développement en lignes de produits logiciels car toute la différenciation des produits au sein de la ligne de produits se base sur elle. Elle devient donc un point clé qu'il faut pouvoir extraire pour comprendre le système dans son environnement.

2 Les systèmes de type web – 2 visions différentes

Il existe une multitude de types de systèmes : standalone, web, embarqués, cloud, bases de données, Operating System ... Ce mémoire s'intéressant plus particulièrement aux systèmes web, il convient de bien fixer le terme qui peut avoir 2 grandes orientations quant au but d'une application web. D'une part, il existe des applications web de type « application Internet » et d'une autre part, celles orientées « application de gestion interne d'entreprise».

1. Les systèmes web orientés Internet (typiquement site web)

Dans le cadre de ce type de systèmes web, 3 remarques peuvent être formulées.

- Le développement web Internet est assez récent, souvent utilisé par des jeunes développeurs non experts et aucune structure dans la programmation n'est encore réellement présente (malgré une nette amélioration). Par exemple, le langage de programmation le plus connu orienté web est le PHP[1] (Hypertext Preprocessor). Le PHP est très permissif, non obligatoirement typé (ce qui lui confère des avantages[2] et inconvénients[3] bien entendu) malgré une nette amélioration depuis PHP V.5 et ne renseigne les problèmes de variabilité qu'au runtime une fois toutes les parties des codes sources interprétées ensemble (il en est de même pour tout code source interprété à la volée).

- Les systèmes de type web Internet sont bien souvent créés sur base de modules développés par différents développeurs, chacun de leur côté, avec leurs niveaux et manières de développer (ex. : WordPress, Joomla, PhpBB ...). Cela crée immanquablement des problèmes de compatibilité.
  Certains systèmes comme WordPress ont mis en place un mécanisme permettant de vérifier la compatibilité entre plugins. Ce mécanisme se base essentiellement sur le retour des utilisateurs et des tests effectués par les développeurs. Cette technique montre bien que la vérification d'un plugin couplé avec un autre est encore très (trop) basique.

  Les systèmes de type web comme WordPress sont donc un bon exemple de système sous forme de ligne de produits logiciels car chaque système ayant sa propre combinaison de plugins (groupables en super plugin) représente un produit et certains plugins ne sont pas compatibles entre eux. Le grand nombre de plugins disponibles (37 699[4]) rend impossible une gestion manuelle.

---

[1] http://php.net

[2] Ces avantages peuvent être un accès plus facile aux jeunes développeurs voulant s'initier à la programmation, mais aussi lorsqu'un changement doit être effectué, il sera plus rapide car aucun type ne doit être géré ...

[3] Ces inconvénients sont aussi multiples, citons le manque de clarté (seul le nom de la variable donne une information sur son contenu), l'impossibilité de connaitre automatiquement et au préalable le type qu'aura une variable à l'exécution et ainsi pré-corriger les éventuelles erreurs avant leurs exécutions ...

[4] Nombre provenant directement du site officiel WordPresse : https://wordpress.org/plugins visité le 07/05/2015 à 11h30.

- Les systèmes web Internet tels que « Amazon » peuvent être un élément clé pour une entreprise. Depuis quelques années déjà, ce genre de site web de vente en ligne s'est fortement développé, entraînant avec lui plusieurs milliers d'emplois à la clé (87 000 employés grâce au e-commerce en 2013 avec une augmentation de 12 000 emplois par rapport à 2012), brassant plusieurs milliards d'euros (51,1 milliards d'euros en 2013 avec une augmentation de +13.5 % par rapport à 2012) (fédération e-commerce et vente à distance 2014). Ces systèmes prennent donc une place importante dans notre société et requièrent de plus en plus de maintenance/améliorations/corrections, ce qui exige une connaissance approfondie du système.

2. Les systèmes web orientés gestion interne d'entreprise

Ces systèmes ne sont pas orientés site Internet mais plutôt gestion interne d'une société. L'informatique étant toujours de plus en plus présente dans le quotidien des sociétés, il est logique qu'elle devienne un point sensible de celles-ci. Comme tout point sensible, elle doit être correctement mise en place et surtout très bien documentée afin d'améliorer les processus de maintenance et d'évolution. En effet, ces systèmes sont sujets à plusieurs évolutions, d'autant plus que leur place ne cesse de grandir au sein de la société.

Prenons l'exemple de l'application « OmniPro[5] » qui est un système web interne permettant la gestion des dossiers des patients d'un hôpital. Au départ, il était une petite application développée par quelques ingénieurs de formation gérants de la société MIMS (novembre 1997). A cette époque, ce système était destiné aux cabinets médicaux, puis, petit à petit, a évolué pour devenir un incontournable dans les hôpitaux belges. A l'heure actuelle, il est décliné en plusieurs produits suivant 3 grands axes (médical, infirmier et organisationnel) dont chacun est disponible avec plus ou moins de modules.
Pour ce genre de système, avoir une documentation précise et complète est donc essentiel. Dans cette optique, les techniques de rétro-ingénierie permettent de donner une aide en proposant des vues sous forme de différents modèles du système.

[5] http://www.mims.be/hospitalier

Chapitre II. Etat de l'art

1 Les lignes de produits logiciels

1.1 Quelques définitions

> « Une **ligne de produits logiciels** est un ensemble de systèmes partageant un ensemble de propriétés communes et satisfaisant des besoins spécifiques pour un domaine particulier.» (ZIADI 2004) (Elkaim. 2001) (L.Northrop 2001)

Une *propriété commune* est une propriété qu'a tout système appartenant à une ligne de produits logiciels. Cette propriété commune est aussi appelée « *commonality* ».
Toute autre propriété non commune (leurs différences) est appelée « *variabilité* ».

> « La **commonality** regroupe l'ensemble des hypothèses qui sont vraies pour tous les produits, membres de la ligne de produits. » (ZIADI 2004) (Lai 1999)

> « La **variabilité** regroupe l'ensemble des hypothèses montrant comment les produits, membres de la ligne de produits, diffèrent. » (ZIADI 2004) (Lai 1999)

> « Un **domaine** est un secteur de métiers ou de technologies ou des connaissances caractérisées par un ensemble de concepts et de terminologies compréhensibles par les utilisateurs de ce secteur. » (ZIADI 2004) (Elkaim. 2001)

1.2 Ses origines

La publication de Parnas (Parnas 1976) mentionne que, à l'origine, lorsqu'un changement survient dans une exigence ou lors de l'évolution (en termes de performance) d'un logiciel, le nombre de versions augmente aussi. Toutes ces versions peuvent être vues comme une famille de produits. Dès lors, la maintenance de ces versions engendre d'énormes coûts financiers. Ce problème est rencontré par beaucoup d'entreprises. Parnas adopte l'idée de considérer la famille de produits comme un tout au lieu de prendre chaque produit séparément donnant ainsi un coût de développement et de maintenance moins élevé.

La publication de Klaus Schmid et al. (Schmid et Verlage 2002) soutient ce principe en mentionnant que les entreprises sont capables de diminuer le délai de commercialisation par un facteur 10, voire plus, si elles utilisent l'approche par ligne de produits logiciels. La Figure 1 (a) extraite de l'article susmentionné montre explicitement la différence d'effort financier entre un développement prenant en compte chaque variante de produit séparément et celui les considérant comme une famille de logiciels[6]. Il en ressort que le coût à l'origine d'un développement par famille est supérieur à celui d'un développement traditionnel. Cependant, plus le

[6] Remarque : le développement utilisé pour ce graphique est celui suivant le pattern particulier « Big Bang » et pas un développement incrémentiel.

nombre de produits augmente plus le coût par produit diminue. Pour en savoir plus, l'article de McGregor (McGregor, et al. 2002) explique qu'une fois la limite de 3 produits (break even) appartenant à la même famille atteinte, le développement par approche de ligne de produits logiciels devient financièrement plus intéressant.

Pour information hors sujet, la publication de Klaus Schmid et al. donne aussi une idée de la différence entre le développement incrémental et le développement suivant le Big Bang pattern Figure 1 (b).

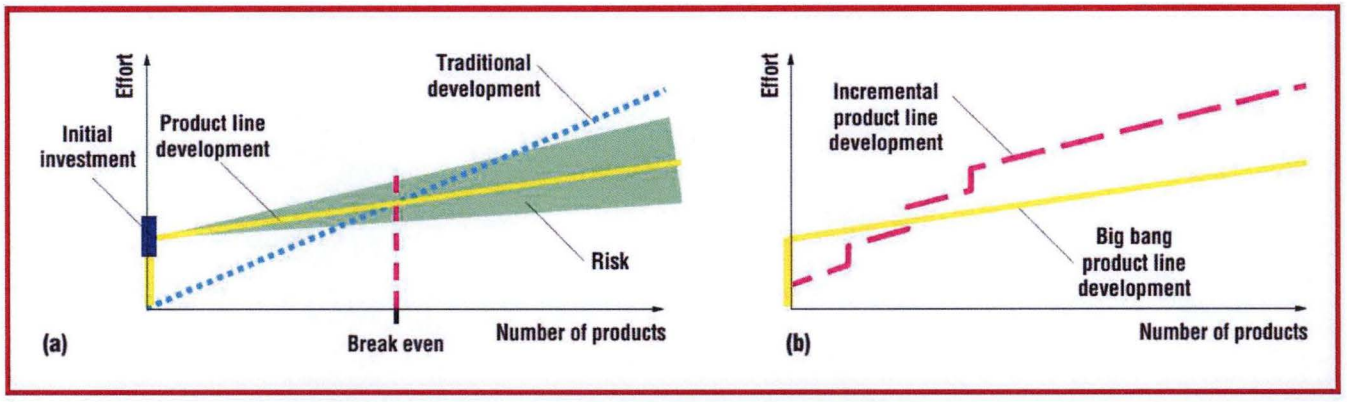


Figure 1 - Comparaison entre l'effort financier d'un développement normal et celui d'une approche en ligne de produits logiciels

1.3 Un paradigme entier

Le *paradigme de lignes de produits logiciels* est directement dérivé des chaînes de production industrielle. Son but principal étant donc de limiter les coûts de construction via le principe de réutilisation d'artéfacts dans le but de créer une série de logiciels tels que décrits précédemment.

Ce paradigme n'est pas nouveau, il est directement inspiré du monde des lignes de productions industrielles. Pour chaque « monde », est repris un exemple ci-après :

- Monde matériel
  Dans le cas d'une chaîne de production de chemises, les différents modules peuvent être la taille (S, M, L, XL, XXXL (USA uniquement)), les manches (longues ou courtes), les boutons de manchettes (1,2 ou 3 boutons), le col (classique, à bouton, mao), le motif (lignes verticales, lignes horizontales, petits carreaux, grands carreaux) ...

- Monde logiciel
  Dans le cas d'une ligne de production d'un logiciel web de type « CMS » (Content Management System), les différents artéfacts (logiciels ou non) peuvent être le template (old school, classique, futuriste), l'espace membre (existant ou non), le forum (existant anonyme autorisé, existant privé, non existant), une newsletter (existante ou non, automatique ou manuelle), le module d'achat en ligne (réservation en magasin ou livraison), le mode de paiement (bancontact, carte de crédit, paiement à la réception, paiement à l'enlèvement, PayPal) ...

Une remarque importante est à ajouter dans ce 2ème monde : la réutilisation des artéfacts s'effectue non seulement dans le code source mais aussi dans l'analyse. Comme cela, toutes les spécifications, la documentation et les différents tests (fonctionnels ou non) sont aussi réutilisés et doivent par conséquent être pensés pour être réutilisables.

Ce paradigme implique des changements importants par rapport au développement traditionnel. Le plus grand des changements réside dans la création d'une *architecture* incorporant l'ensemble des artefacts créant ainsi une ligne de produits logiciels.
Cette architecture se confronte à un problème de taille majeure : la *compatibilité entre les différents modules*. Une mauvaise gestion de cette compatibilité pourrait amener à des systèmes totalement absurdes :

Tableau 1 - Absurdités possibles dans une chaîne de production

Monde matériel	Monde logiciel
« Un véhicule à essence muni d'un filtre à particules. » Ce cas aurait dû techniquement être une exclusion de composants.	*« Une version limitée en lecture seule avec un module d'enregistrement. »* Ce cas aurait dû techniquement être une exclusion de composants.
« Un distributeur de billets sans module de réception de billets. » Dans ce cas, ce sont des composants manquants qui mènent à l'absurdité.	*« Un site Internet demandant une connexion obligatoire avant un achat sans module de connexion. »* Dans ce cas, ce sont des composants manquants qui mènent à l'absurdité.
« Une lampe de poche munie d'un câble électrique. » Ce cas aurait dû techniquement être une exclusion de composants.	*« Un forum privé avec la présence d'un utilisateur par défaut et anonyme. »* Ce cas aurait dû techniquement être une exclusion de composants.

Afin de combler ce potentiel problème, l'ingénierie logicielle se voit scindée en 2 parties (Figure 2 ci-dessous).

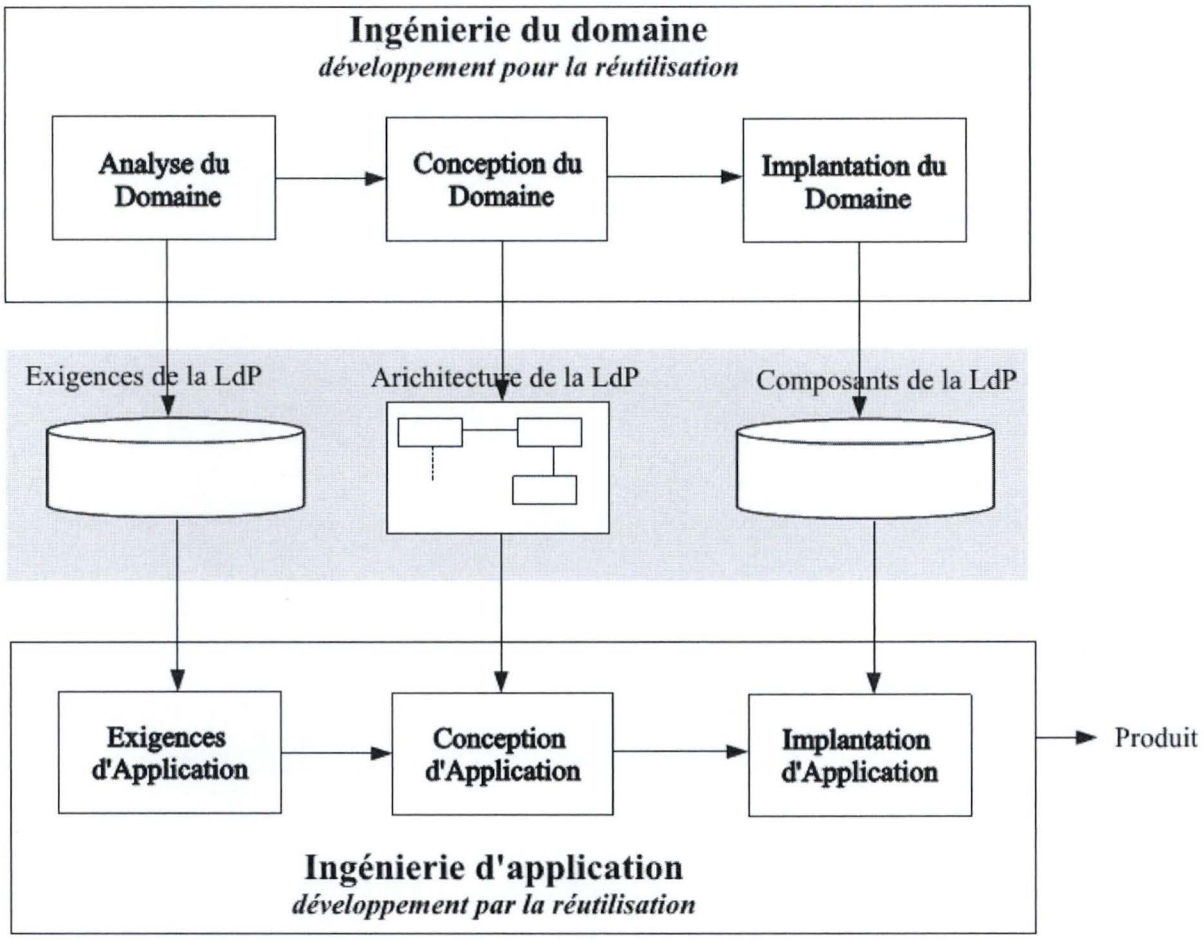


Figure 2- Ingénieries sous le paradigme des lignes de produits logiciels (ZIADI 2004)

1. Une ingénierie du domaine

   L'ingénierie du domaine se focalise sur le développement et la documentation permettant de développer un logiciel appartenant à une ligne de logiciels.
   Cette partie est constituée de trois activités.

   I. L'analyse

   Cette activité a pour but la détermination et l'identification des comonalités et variabilités. Un modèle des plus connus dans cette activité est le modèle « feature model » (FM). Ce modèle a été introduit la première fois dans la méthode FODA (Feature-oriented domain analysis) par Kang (Kang, et al. 1990)

   II. La conception

   Cette activité a pour but la création de l'architecture logicielle «générique».

   III. L'implémentation

   Cette dernière activité a pour but l'implémentation de l'architecture de l'activité « conception » sous forme de composants réutilisables par l'ingénierie d'application (dans le but de la construction de chaque produit).

2. Une ingénierie d'application
Se basant sur le résultat de l'ingénierie du domaine, l'ingénierie d'application s'occupe de la construction d'un produit particulier faisant partie d'une ligne de produits logiciels (étapes normales de la production d'un logiciel tout en prenant compte des résultats de l'ingénierie de domaine).

1.4 La variabilité – un rôle central

La variabilité étant un concept clé de ce paradigme, elle devient donc un point sur lequel il faut s'attarder. Ce mémoire ayant pour but essentiel la rétro-ingénierie de la variabilité dans les systèmes web sous forme de lignes de produits logiciels, il est donc logique de décrire précisément « ce qui se cache » derrière cette expression.

1.4.1 Qu'est-ce que la variabilité ?

La variabilité au sein des lignes de produits est, suivant sa définition (voir section 1.1 p.16), une représentation des différences entre les différents produits. Elle doit être gérée tout au long du cycle de vie d'un logiciel depuis la spécification des exigences jusqu'aux tests finaux en passant par l'architecture, le code source du système, ...

La variabilité est habituellement représentée sous la forme de *points de variations* et *variantes* (HEYMANS et TRIGAUX 2003).

- Un point de variation localise une variabilité et ses liaisons en décrivant plusieurs variantes.
- Une variante représente une alternative (de design) dans le but de réaliser une variabilité particulière et de la lier de manière concrète.

La variabilité dans les lignes de produits logiciels est un point difficile à gérer car, entre autres, chaque produit et famille de produits peut évoluer et changer (que cela soit l'apparition d'une release ou un changement dans la variabilité) donnant ainsi naissance à deux concepts : « Variation in time » (concept mentionnant le fait qu'il puisse exister différentes releases de produits) et « Variation in space » (concept mentionnant le fait qu'en même temps, 2 produits peuvent contenir des variantes égales étant différemment implémentées).

1.4.2 Vers une classification de la variabilité

Un schéma de classification des différentes variabilités observables dans les systèmes est proposé par Felix Bachmann et Len Bass (Bachmann et Bass 2001) suivant 6 critères.

- La variabilité dans les fonctions : ce critère mentionne le fait qu'une fonction particulière peut exister dans certains produits mais pas nécessairement dans tous.

- La variabilité dans les données : ce critère mentionne qu'une structure de données particulière peut être différente d'un produit à un autre.

- La variabilité dans les contrôles de flux : ce critère signifie qu'un flux de contrôle d'interaction (l'ordre dans lequel les éléments d'un logiciel comme les appels des fonctions, les instructions ... s'exécutent) peut être différent d'un produit à un autre.

- La variabilité dans la technologie : ce critère mentionne une différence concernant les plateformes sur lesquelles les produits doivent fonctionner (ex. : le système d'exploitation, le hardware, les dépendances avec des artéfacts tels que le sont les middlewares ...). Ces plateformes sont potentiellement fortement différentes au niveau technique.

- La variabilité dans la qualité du produit : ce critère mentionne la qualité que doit avoir un produit en termes de sécurité, performance, disponibilité, modificabilité ...

- La variabilité dans l'environnement d'un produit : ce critère mentionne qu'un domaine de produit peut imposer des exigences bien précises.

Il existe d'autres classifications telles que décrites par Günter Halmans et Klaus Pohl (Halmans et Pohl 2003)se basant sur 2 concepts : la « Etential variability » (concept déterminant les exigences fonctionnelles et non fonctionnelles à implémenter du point de vue client/utilisateur, ce qui définit le « quoi ».) et la « Technical variability » (concept qui détermine comment implémenter le système, ce qui définit le « comment »).

1.4.3 Quand la variabilité est-elle implémentée ?

Il existe 3 moments où la variabilité peut-être implémentée/fixée. Le choix d'un moment a bien évidemment un impact sur toute l'architecture dans le sens où celle-ci devra prendre en compte ce moment (ex. : doit-il mettre en place un mécanisme d'héritage ou de paramétrisation de l'architecture ?).

1. A la compilation
   Les différentes features sont choisies lors de la compilation (via le choix des packages). Ce choix peut être appliqué si la recherche de l'efficacité (en temps et espace disque) est de mise, mais aussi si les features aboutissent à différents produits ou encore si le produit a peu de chance d'être modifié par la suite.

2. Au chargement
   Les features sont choisies au chargement du logiciel (elles ne sont plus changeables une fois le logiciel en cours d'exécution). Typiquement, ce genre de choix s'effectue sur les features qui sont en relation avec l'environnement du logiciel (ex. : le système d'exploitation) et le type de mission pour laquelle le logiciel est lancé (ex. : les systèmes d'armement).

   Remarque : l'architecte doit mettre en place un paramètre permettant de choisir les types de features qui seront chargées au démarrage du logiciel.

3. Pendant l'exécution
   Les features peuvent être choisies pendant l'exécution du logiciel. Cette modification peut être manuelle ou suivant un mécanisme de changement automatique (ex. : les logiciels embarqués dans les véhicules automobiles pourraient charger des modules suivant le type de conduite que le conducteur souhaite. Le changement d'un « mode normal de conduite » à « conduite sur glace » ou « conduite sportive » peut être vu comme un changement de feature sans arrêter le système).

Le choix du moment n'est pas un choix unique et fixe. Il est dès lors possible d'implémenter certaines features à la compilation, d'autres au chargement du logiciel et d'autres à l'exécution de celui-ci.

1.4.4 En savoir plus sur les feature models

Comme mentionné précédemment, le modèle très souvent utilisé quant à la gestion de la variabilité est le « feature model ». Kang et al. (Kang, et al. 1990) définissent le feature model comme :

> « Un feature model représente les features standards d'une famille de systèmes dans un domaine et les relations entre elles.
> [...]
> La relation structurelle est une constitution de features, ce qui représente un groupe logique de features, présentant un intérêt.
> [...]
> Les features alternatives ou optionnelles doivent être indiquées dans le modèle.
> [...]
> Toutes les features doivent être distinctivement nommées et leurs définitions devraient être incluses dans le dictionnaire des termes du domaine. »
>
> (traduit par PATINY Mathieu)

Kang fait également remarquer que la sémantique existante entre les features n'est pas exprimée dans les feature diagrams. Il entend par là, que toutes les features alternatives et optionnelles qui ne peuvent pas être sélectionnées une fois la feature choisie doivent être marquées comme « exclusives entre elles ». Les autres features qui doivent être sélectionnées doivent être marquées comme « requises ». Ces marquages établissent ensemble les « règles de compositions » et forment ainsi la

sémantique manquante. Le choix de marquage de ces features n'est évidemment pas aléatoire ni automatique, il provient d'un choix totalement objectif par rapport aux demandes du client.

La documentation d'un feature model.

La *documentation d'un FM* reprend un ensemble d'informations précises.

- Le diagramme reprenant la structure hiérarchique des features avec les mentions concernant celles optionnelles et alternatives,
- La définition de chaque feature (un exemple de la forme que doit prendre une définition de feature est disponible en annexe I.),
- Les règles de composition des features

L'usage principal des feature models.

Les FMs ont un but principal de communication entre les développeurs et les utilisateurs.
Du point de vue développeurs, ces modèles permettent d'exprimer quels sont les artéfacts qui doivent êtres paramétrés (ce qui a un impact significatif sur toute l'architecture) et comment ils doivent êtres paramétrables.
Du point de vue des utilisateurs, ceux-ci pourront ainsi découvrir quels sont les features de base, celles ajoutables ainsi que le moment où ils peuvent les choisir (voir les 3 types d'implémentations de la section 1.4.3).

D'un point de vue pratique, ce genre de modèle est justement approprié dans un processus d'aide à la vérification de compatibilité entre plugins implémentés comme mentionnés dans le chapitre I part 2 point 1 (système web Internet ayant un système de plugins). Le client qui installe un nouveau plugin dans son système pourrait directement savoir avec quel module il est ou n'est pas compatible. En poussant l'utopie à son plus haut niveau, un tel modèle pourrait faire une sélection automatique des plugins ajoutables sur base du système déjà présent (modélisable via des techniques de rétro-ingénieries de système implémenté). Ce processus pourrait ainsi fixer les features déjà utilisées et en déduire celles autorisées.

Feature model VS feature diagram.

La littérature associe souvent les 2 termes : FM à feature diagram (Acher, et al. 2012). En réalité, cette association n'est pas tout à fait correcte. Il serait plus correct de dire qu'un FM peut contenir plusieurs features diargams en ajoutant des formules propositionnelles.

Pour rappel, les formules propositionnelles sont des formules mathématiques composées de propositions. Une proposition étant « un élément » considéré comme vrai et totalement atomique (non décomposable). Un exemple de formule pourrait

être A ∧ (B ∨ C) ⇒ D. Pour en savoir plus ces formules liées à ce genre de modèle il est conseillé de lire la publication « Feature Models, Grammars, and Propositional Formulas » (Batory 2005).

Pour permettre une bonne compréhension, il faut tout d'abord définir ces 2 termes précisément :

> « Un **features diagram** est défini comme un ensemble (G, r, E_{mand}, G_{mutex}, G_{xor}, $G_{or,}$, BI, I, EX) composé des éléments suivants :
> - L'arbre G = (F,E,r) où :
> - F est un ensemble fini de features,
> - E est un ensemble fini de liaisons entre features (E ⊆ F x F),
> - r est le neod racine.
> - E_{mand} ⊆ E est un ensemble de liaisons qui définissent les features obligatoires.
> - G_{mutex} ⊆ P(F) x F, G_{xor} ⊆ P(F) x F, G_{or} ⊆ P(F) x F définissant des groupes de features et sont des ensembles de paires de features enfants avec leurs features parents communs.
> - BI : Un ensemble de contraintes de bi-implication de la forme A ⇔ B,
> I : Un ensemble de contraintes d'implication de la forme A ⇒ B,
> EX : Un ensemble de contraintes d'exclusion sous la forme A ⇒ ¬B (A ∈ F and B ∈ F). »
>
> (traduit par PATINY Mathieu)

> « Un **feature model** est un tuple (FD, ψ_{cst}) où :
> - FD est un feature diagram.
> - ψ_{cst} est un ensemble de formules propositionnelles appliquées sur l'ensemble des features F de DF. »
>
> (traduit par PATINY Mathieu)

L'application des formules propositionnelles se fait visuellement via les « cross-tree constraints » (CTCs). Ces annotations sont des annotations uniquement visuelles, elles ne font qu'exprimer les formules au sein d'un FM sans apporter une quelconque information en plus.

Pour illustrer la théorie, Acher et al. proposent une illustration d'un FM (Figure 4) complet provenant d'une table de données (Figure 3)

Identifier	License	Language	Storage	LicenseCostFee	RSS	Unicode
Confluence	Commercial	Java	Database	US10	Yes	Yes
PBwiki	Nolimit	No	No	Yes	Yes	No
MoinMoin	GPL	Python	Files	No	Yes	Yes
DokuWiki	GPL2	PHP	Files	No	Yes	Yes
PmWiki	GPL2	PHP	Files	No	Yes	Yes
DrupalWiki	GPL2	PHP	Database	Different Licences	Yes	Yes
TWiki	GPL	Perl	FilesRCS	Community	Yes	Yes
MediaWiki	GPL	PHP	Database	No	Yes	Yes

Figure 3 - Illustration de la théorie des feature models - table de données (Product Description)

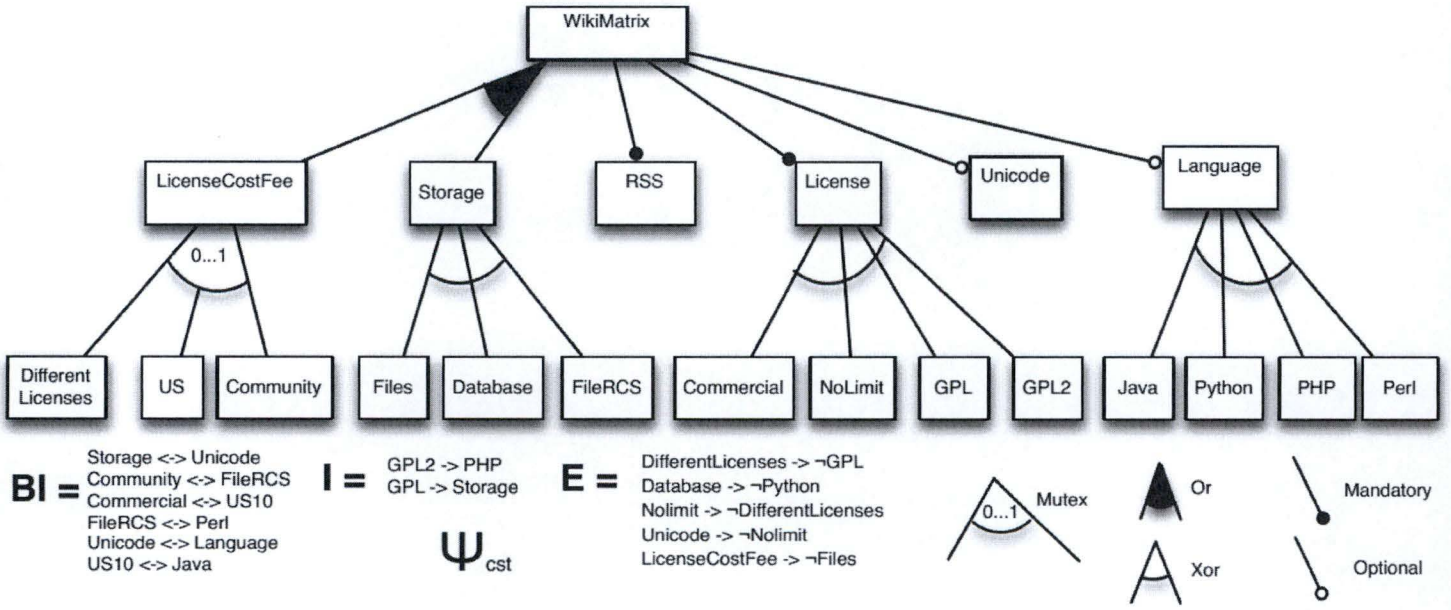


Figure 4 - Illustration de la théorie des feature models - feature model

Un feature model par l'exemple.

Ce modèle correspond à une représentation sous la forme d'arbre de toutes les comonalités et (surtout) les variabilités dont les relations sont visuellement annotées par des contraintes (« cross-tree constraints ») (Cuevas 2007). Il existe 2 types de contraintes reprises dans le Tableau 2 et Tableau 3 (les exemples sont liés à la Figure 5 p. 28) :

Tableau 2 - Légende des contraintes sur feature models : parent – enfants

Entre : parent – enfants
• Obligation *La feature doit obligatoirement* être présente dans toutes les instances d'un produit contenant la feature parent. Ex. : Une chemise doit obligatoirement avoir une taille, un type de manche et un logo.
• Optionnel *La feature n'est pas obligatoire* pour les instances des produits contenant la feature parent. Ex : Une chemise n'est pas obligée d'avoir un logo.

Tableau 3 - Légende des contraintes sur feature models : parent – enfants

Entre : enfants
• And *Toutes les features enfants doivent obligatoirement* être présents dans toutes les instances d'un produit contenant la feature parent.
• Or *Une ou plusieurs features enfants* peuvent être présentes dans toutes les instances d'un produit contenant la feature parent. Ex. : Une chemise peut avoir un logo sur le torse et/ou dans le dos.
• Xor Seule *une feature enfant au maximum* peut être choisie parmi les enfants dans toutes les instances d'un produit contenant la feature parent. Ex. : Une chemise doit avoir un type de manche courte ou longue mais pas les 2.

Un autre exemple issu de la publication de Benavides et al. (David Benavides 2010) reprenant ces types de contraintes ainsi que leurs formules propositionnelles associées se trouve en Annexe II.

Certains auteurs (Czarnecki (K. Czarnecki 2005) et Riebisch (M. Riebisch 2002)) proposent une extension à ces contraintes : les cardinalités.

- Cardinalité de feature : modélisé par un d'intervalle noté [n..m] et représentant le nombre minimal (n) et maximal (m) d'instances de features autorisables dans le produit.
  Ex. : dans la Figure 6, le CMS peut accepter jusqu'à 3 types d'instances de cartes de crédit.

- Cardinalité de groupe : modélisé par un intervalle noté <n..m> et représentant le nombre minimal (n) et maximal (m) d'enfants qu'un produit peut contenir si la feature parent est présente (par défaut <1..1>).
  Ex. : dans la Figure 6, le CMS peut accepter des paiements en ligne. Si c'est le cas, il peut proposer entre 1 et 2 types de systèmes de paiement en ligne.

Deux autres contraintes peuvent être aussi appliquées (David Benavides 2010) entre features qui n'ont pas nécessairement une relation de type parent-enfants ou entre enfants :

- Inclusion entre 2 features précises : A - - → B. Ce symbole mentionne le fait qu'une feature (A) requière une autre feature (B).

- Exclusion entre 2 features précises : A ← - → B. Ce symbole mentionne le fait qu'une feature (A) exclut une autre feature (B).

Ce modèle peut s'appliquer tout aussi bien à des produits du monde matériel comme des produits du monde logiciel. La Figure 5 ci-après représente un exemple de modélisation d'une ligne de production de chemises (monde matériel). La Figure 6 quant à elle, représente un exemple de modélisation d'une ligne de production d'un CMS standard (monde logiciel).

Dans le cas de la Figure 5, voici quelques produits pouvant correspondre à ce FM :

- 1 chemise de taille XL à manches courtes contenant un logo sur le dos et le torse sans aucun motif.
- 1 chemise de taille XL à manches courtes contenant un logo sur le dos et le torse avec un motif en ligne horizontale.
- 1 chemise de taille XL à manches longues contenant un logo sur le torse avec un motif en forme de carreaux.

Contre-exemple de produits ne pouvant pas exister :

- 1 chemise de taille XL sans manche avec un logo sur le torse et sans motif. Cette chemise serait incompatible avec ce modèle car toute chemise exige un type de manche dans le modèle.

L'interprétation du modèle est exactement la même dans le monde matériel que dans le monde logiciel.

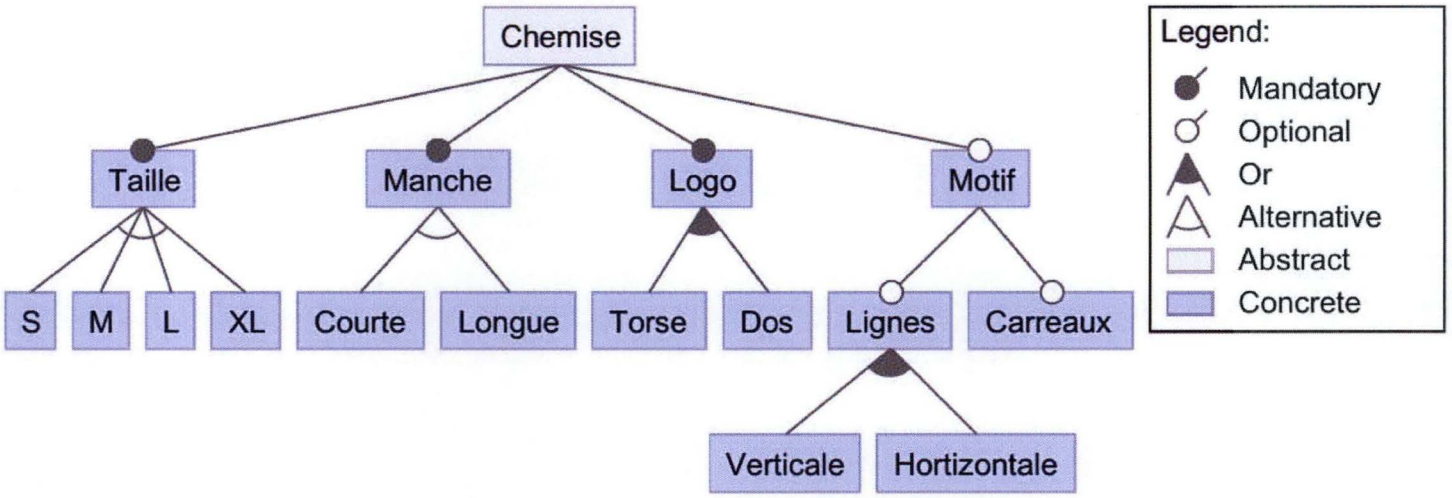


Figure 5 – Exemple de feature model – monde matériel : "la ligne de produits de chemises"

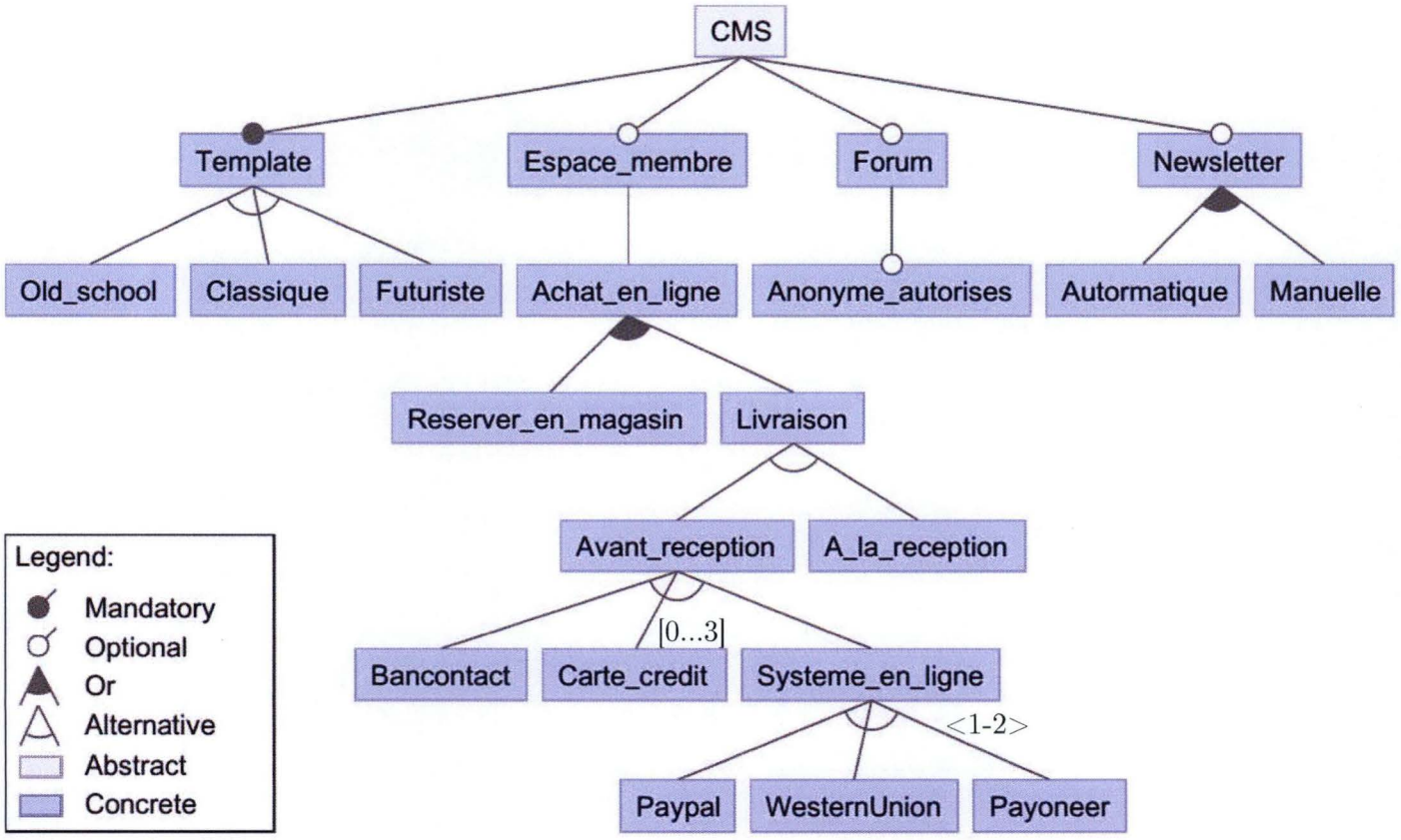


Figure 6 - Exemple de feature model – monde logiciel : "la ligne de produits de sites Internet avec possibilité de vente en ligne via CMS"

Plusieurs feature models pour une même sémantique.

Une remarque importante doit être mentionnée : pour une sémantique unique, il peut exister plusieurs feature models. Un exemple tout simple permet de s'en rendre compte :

Dans le 1er cas, tous les produits possibles sont composés des features : B et B+C.
Dans le 2e cas, les produits possibles sont totalement identiques : B et B+C.
Dès lors, il ne faut pas partir du principe qu'une technique de rétro-ingénierie produira toujours le même FM. Il existe des outils permettant la comparaison de FMs.

Manipulation de feature models.

Benavides (David Benavides 2010) propose 3 définitions sur lesquelles la manipulation de FM peut se baser :

> « Pour un FM donné avec un ensemble de features F, une **configuration** est un 2-tuple sous la forme (S, R) tel que S, R ⊆ F et où
> - S est l'ensemble de features devant être sélectionnées,
> - R est l'ensemble de features devant être supprimées tel que S R = ∅ »
>
> (traduit par PATINY Mathieu)

> « Une configuration est appelée **configuration partielle** si S ∪ R ⊂ F »
> « Une configuration est appelée **configuration complète** si S ∪ R = F »
>
> (traduit par PATINY Mathieu)

La littérature relate une multitude d'opérations plus ou moins complexes applicables sur les feature models. L'article de Benavides reprend une liste exhaustive des fonctions réalisables. Parmi celles-ci, les plus courantes sont :

- Déterminer si un FM est vide : ce processus permet, depuis un FM en entrée, de déterminer s'il est vide ou non. Un FM est vide dans le cas où aucun produit ne peut en être déduit (ex. : si les contraintes empêchent toute combinaison de features).

- Déterminer si un produit est valide : en partant d'un FM et un produit donné (ex. : sous forme d'un ensemble de features), ce processus permet de déterminer si le produit fait partie des produits composables depuis le FM.

- Valider une configuration partielle : depuis un FM et un ensemble de configurations partielles, le processus retourne si l'ensemble de configurations contient des contradictions (ex. : 2 configurations contenant des features exclues par des contraintes).

- Générer tous les produits : depuis un FM, le processus retourne tous les produits valides dérivables.

- Nombre de produits : depuis un FM, le processus retourne le nombre de produits dérivables.

- Filtrer les produits : en partant d'un FM et une configuration (partielle ou complète), le processus retourne l'ensemble des produits de la configuration ne pouvant être produit par le FM.

- La détection d'anomalies : de nombreuses anomalies peuvent être détectées sur base d'un FM (ces anomalies peuvent, par exemple, être présentes dans un FM provenant d'une technique de rétro-ingénierie ayant une certaine marge d'erreur) :

    - Features mortes : une feature est considérée comme morte si elle ne peut être incluse dans un produit dérivable du FM. Ce genre de cas peut survenir dans les cas où des contraintes (d'exclusion) empêchent l'utilisation d'une feature.

    - Features mortes conditionnelles : ce genre de feature est une feature morte sous certaines conditions, comme le fait d'avoir sélectionné une autre feature particulière présentant une contrainte. Elles sont considérées comme des contradictions. Ce genre de contradiction peut apparaître lors de l'évolution d'un FM au cours du temps (maintenance, amélioration ...). Un exemple pourrait être serait le suivant :

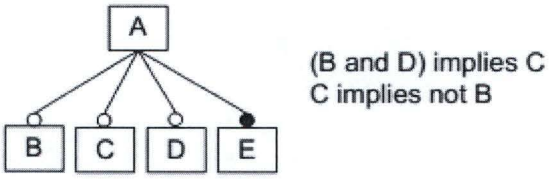


Figure 7 – Features mortes conditionnelle (David Benavides 2010)

    - Mauvaises cardinalités : si la présence d'une contrainte de cardinalité n'est pas réalisable ou provoque une absurdité. Dans l'exemple suivant, B et D s'excluent mutuellement mais une contrainte de cardinalité les autorise simultanément.

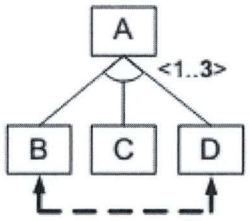


Figure 8 - Mauvaises cardinalités (David Benavides 2010)

- Redondance : de la redondance sémantique peut apparaître dans les FMs. Dans l'exemple suivant (les flèches grisées donnent lieu aux redondances) :
  (a) B et C sont déjà mutuellement exclus par la « cross-tree contstraint »,
  (b) Une redondance existe dans le fait que C est toujours présent, la contrainte d'implication B implique C est donc redondante,
  (c) Une redondance existe entre 2 implications (si B implique C, D étant enfant obligatoire de B, C est d'office impliqué).

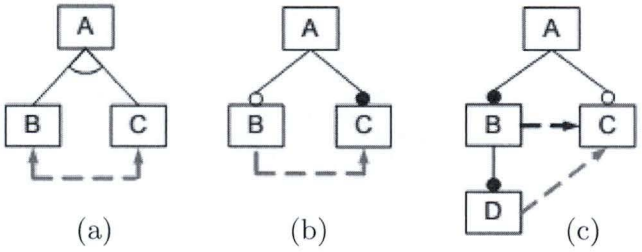


Figure 9 - redondance (David Benavides 2010)

- Détermination des relations entre FMs : entre plusieurs FMs, il peut exister des relations potentiellement intéressantes pour les processus :
  - Reconstruction : leurs structures sont différentes mais ils représentent la même suite de produits (voir précédemment).
  - Généralisation : une généralisation existe entre 2 FMs (FM2 généralise FM1) si tous les produits déductibles de FM1 sont contenus dans les produits déductibles de FM2.
  - Spécialisation : une spécialisation existe entre 2 FMs (FM1 spécialise FM2) si tous les produits déductibles de FM1 sont contenus dans les produits déductibles de FM2.

Les outils de gestion de FMs comme « FeatureIDE » et « FAMILIAR » proposent des fonctionnalités comme celles-ci permettant ainsi une manipulation automatisée de FMs complexes.

Dans le cadre d'une rétro-ingénierie de FMs, des manipulations sur ceux-ci doivent être effectuées. Les processus décrits précédemment permettent de mieux gérer les FMs afin de détecter, par exemple, immédiatement, toute anomalie dans un FM extrait du système ou encore dans la vérification de ce FM avec les produits théoriquement implémentés.

Les tests automatisés et correction de feature models.

Lorsqu'un FM est déduit depuis un artéfact d'un système, il se peut qu'il reste encore des erreurs dans celui-ci. Afin de résoudre au mieux ces erreurs, Henard et al. (Henard, et al. 2013) proposent un processus empirique automatisé (les processus existants alors, ne l'étaient qu'en partie seulement) permettant de faire correspondre le modèle de variabilité au système analysé. Ce processus pourrait être utile dans le cadre de la rétro-ingénierie.

Le processus proposé fonctionne en 2 parties représentées dans la Figure 10.

1. Tester le FM

   Cette première partie permet la mise en évidence de problèmes/erreurs du FM. A ces fins, 2 étapes sont nécessaires :

   I. Evaluer la cohérence du FM
   Cette première étape permet l'évaluation de la cohérence du FM par rapport aux configurations valides du système. Partant du principe qu'il existe ou qu'il y a moyen d'obtenir les configurations actuelles du système, le FM est évalué sur base des contraintes existant dans ce FM et qui ne sont pas compatibles avec les configurations existantes (EWC). L'ensemble de ces contraintes est ensuite retourné tant qu'il existe des configurations non compatibles avec le FM (SCF).

   II. Evaluer les configurations générées par la FM
   Cette seconde étape considère le sens inverse : elle part du FM et génère les configurations aléatoirement et valides pour ce FM. Sur base d'un oracle défini par le testeur (cet oracle devant représenter les actions que ce dit testeur effectuerait), chaque configuration est notée comme compatible ou non avec le système. Les règles d'oracle qui échouent sont retournées (ORF) tant qu'il existe des configurations générées non compatibles avec cet oracle (GCF)

2. Corriger le FM

Sur base des erreurs mises en avant dans l'étape 1 et d'un FM générique à n features et m clauses/contraintes encodées en forme normale conjonctive (CNF) où chaque contrainte C est une disjonction de k ($\leq$ m) features sélectionnées ou non sélectionnées dans le FM d'origine ($\bigwedge_{i=1}^{n} C_i$), une suite de 3 opérations est effectuable dans le but de les corriger :

- Altération
  Sur base des mauvaises contraintes de l'étape 1 (EWC), une contrainte est sélectionnée aléatoirement et niée.

- Suppression
  Sur base des mauvaises contraintes de l'étape 1 (EWC), une contrainte est sélectionnée dans les EWC et supprimée.

- Insertion
  Sur base des règles d'oracle (ORF) de l'étape 2, une contrainte est ajoutée au FM.

Après avoir effectué une de ces 3 opérations, un FM' est créé. Le FM d'origine est ensuite comparé au FM'. Cette comparaison se base sur le nombre d'incohérences :

- S_1 : #EWC,
- S_2 : #SCF,
- S_3 : #ORF,
- S_4 : #GCF.

En considérant que S_1, S_2, S_3 et S_4 sont ceux appartenant au FM d'origine et S'_1, S'_2, S'_3 et S'_4 sont ceux de FM', FM est remplacé par FM' ssi $\left[(\sum_{i=1}^{4} S'_i < \sum_{i=1}^{4} S_i) \wedge (S'_1 \leq S_1 \wedge S'_2 \leq S_2 \wedge S'_3 \leq S_3 \wedge S'_4 \leq S_4\right] \vee (S'_2 < S_2 \wedge S'_4 \leq S_4) \vee (S'_2 \leq S_2 \wedge S'_4 < S_4)$.

Cette condition permet d'éviter de remplacer FM par FM' s'il provoque un impact négatif sur les autres ensembles. Ce processus bouclera tant qu'il existe des incohérences et ce, jusqu'à ce que le critère de qualité voulu soit atteint (si ce critère est la correction totale du FM, il se peut que des boucles infinies surviennent).

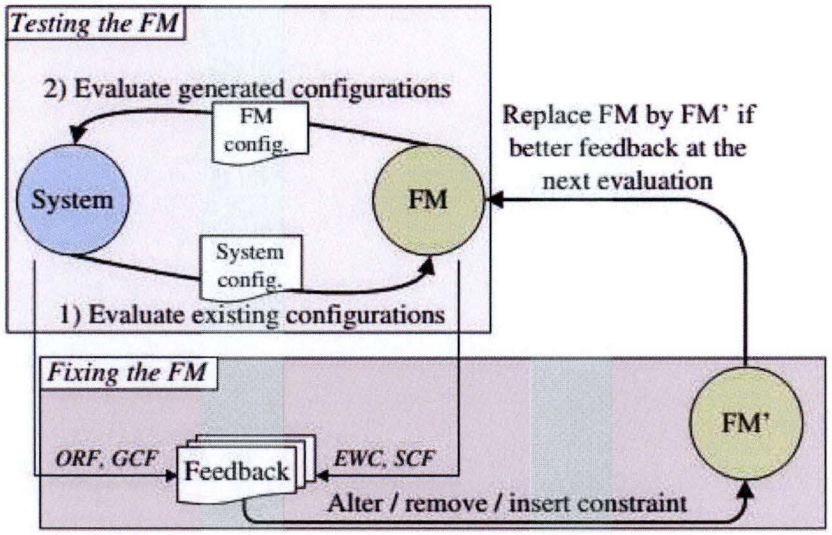


Figure 10 - Processus de correction d'un feature model par la technique de Henard et al. (Henard, et al. 2013)

2 La rétro-ingénierie

2.1 Ses origines

L'origine du terme vient du monde de l'analyse de hardware dans lequel le déchiffrage du design d'un produit depuis des produits finis est à la tendance (Elliot J. Chikofsky 1990).

Les techniques de rétro-ingénierie sont aussi régulièrement appliquées dans le but d'améliorer son propre produit ou encore, utilisées afin d'analyser les produits des concurrents. Dans certains cas plus délicats comme les cas militaires et de sécurité nationale, ce sont les produits des adversaires qui sont analysés (un des plus grands exemples du siècle dernier est le reverse-engineering appliqué sur la machine « Enigma » en fin 1932 par Rejewski puis par Turing).

Les articles publiés respectivement par Rekoff (Rekoff 1985) et Chikofsky (Elliot J. Chikofsky 1990) donnent une définition précise de ce qu'est la rétro-ingénierie hardware et logicielle :

> « La **rétro-ingénierie hardware** est le processus de développement d'une série de spécifications pour un système hardware complexe via « l'examination » ordonnée d'un exemple de ce système.
> [...]
> Sans aucune connaissance de la modélisation d'origine
> [...]
> Dans l'objectif de créer un clone du système hardware original »
>
> (traduit par PATINY Mathieu)

> « La **rétro-ingénierie logicielle** est le processus d'analyse d'un système sujet dans le but :
> - d'identifier les composants d'un système ainsi que leurs relations
> - de créer une représentation d'un système sous une autre forme ou à un niveau d'abstraction supérieur. »
>
> (traduit par PATINY Mathieu)

En partant de ces principes hardware et en les appliquant au software, il est possible d'extraire une compréhension basique de son système et de sa structure. Alors que la rétro-ingénierie appliquée à l'origine l'était traditionnellement dans un but de copie appliquée aux logiciels, elle permet actuellement une aide à la *maintenance*, un *renforcement* ou un *support* en procurant un modèle de design.

L'organisme privé « American National Standards Institute » (ANSI) définit la maintenance de logiciel comme suit :

> « La **maintenance logicielle** est la modification d'un logiciel après sa mise en production dans le but de corriger les fautes, d'améliorer sa performance ou autres attributs, ou adapter le produit à un environnement changé »
>
> (traduit par PATINY Mathieu)

Etant donné que, habituellement, l'équipe s'occupant de la maintenance du système n'étant pas la même que celle s'occupant de son développement, cette première équipe prendra énormément de ressources pour comprendre le système avant de commencer sa maintenance. C'est dans ce but que les outils de rétro-ingénieries permettent de faciliter la tâche.

C'est d'ailleurs dans ce cadre que Rekoff positionne les techniques de rétro-ingénierie dans le processus de maintenance logiciel.

2.2 Ses objectifs

Maintenant que la rétro-ingénierie fait partie intégrante du processus de maintenance, il intéressant de mentionner quels sont les objectifs de la technique et d'en retirer quel est son rôle dans ce processus.

Il existe 2 grands métas-objectifs à la technique.

1. *Identifier les différents composants* du système analysé ainsi que les relations entre eux.

2. *Création d'une représentation* d'un système sous une autre forme ou à un niveau d'abstraction plus élevé (remarque : la technique ne permet pas d'avoir un niveau d'abstraction moins élevé. Si cela avait été le cas, ce processus ne serait pas un processus de rétro-ingénierie mais un processus de dérivation dans un but de création d'un système).

Cependant, le principe de rétro-ingénierie n'a pas pour objectif l'amélioration d'un artéfact ni la création d'un nouveau système basé sur un ancien car c'est un processus dit « d'examination ». Il est entendu par là, un processus qui examine un système sans apporter un quelconque changement.

En rentrant plus en profondeur dans les objectifs, Chikosfky en propose 6 clés précis de la rétro-ingénierie :

- Faire face à la complexité
  Dans les gros systèmes, la complexité et les relations entre composants peuvent être très grandes. Un processus automatique (ou le plus automatique possible) tel que faisable avec la rétro-ingénierie permet d'extraire de l'information pertinente afin d'automatiser un support à cette complexité.

- Générer des vues alternatives
  Le processus de rétro-ingénierie facilite grandement la génération ou régénération d'une représentation graphique du système. Ces vues peuvent s'ajouter aux vues déjà proposées dans l'analyse afin de les compléter et d'apporter d'autres perspectives.

- Récupération d'information perdue
  Dans la vie d'un système, de multiples modifications sont effectuées sur celui-ci sans toujours être totalement documentées (et de moins en moins jusqu'aux designs de haut niveau). Dans ce cas, la rétro-ingénierie permet, en tant que processus de « rattrapage », de fournir une documentation prenant compte des modifications non totalement documentées.

- Détecter les effets de bords
  Depuis le design initial, les multiples modifications apportées au système peuvent mener à des ramifications et apparition d'effets de bords ayant un potentiel impact sur les performances du système. Dans ce cas, la rétro-ingénierie produit différentes vues permettant ainsi d'aider à la détection d'anomalies (avant que l'utilisateur final ne la reporte comme un bug).

- Synthétiser une abstraction de plus haut niveau
  La rétro-ingénierie permettant la génération d'une vue de plus haut niveau, cette dernière pourrait être une vue alternative à celles existantes. Ceci dit, ce processus est rarement totalement automatique. Les experts du système étudié jouent un rôle important dans cette génération de haut niveau.

- Faciliter la réutilisation
  Depuis plusieurs années, le monde du logiciel suit un mouvement qui tente de maximiser la réutilisation de logiciels. Dans ce cadre, le processus peut aider à la détection de composants potentiellement réutilisables d'un système existant.

2.3 Un effet sur le coût du logiciel

Dans le cycle de vie d'un système, le temps passé à apprendre comment fonctionne l'implémentation d'un logiciel (dans le but d'une maintenance, évolution, ...), est du temps perdu et indirectement reporté sur le coût.
Le processus de rétro-ingénierie permet de réduire (voir objectifs décrits précédemment) ce temps et donc l'impact financier qui en découle.

2.4 Deux sous catégories

Il existe 2 grandes sous-catégories à la rétro-ingénierie. Ces dernières sont la « redocumentation » et le « Design recovery » :

❖ Redocumentation
Le but de la redocumentation est de créer une représentation sémantique avec le même niveau d'abstraction que l'artéfact étudié. Le résultat de cette sous-catégorie propose une vue alternative à celles existantes proposable. Cette nouvelle vue doit être totalement orientée : documentation pour un humain.
Ainsi, certains outils tels que les suivants permettent de visualiser les relations entre les composants du système analysé (ainsi que clairement visualiser les chemins qui les relient).

- les « pretty printers » : applications permettant une mise en forme plus élégante de texte. Par exemple, les applications permettant une mise en forme/style sur un code source obfusqué.

- Les générateurs de diagrammes : générateurs permettant la création de diagrammes directement depuis le code source d'une application offrant ainsi une visualisation de la structure et du flux de contrôle de ce code source.

- Les générateurs de « cross-reference listing »: générateurs proposant un mapping entre le code source et une liste de variables, nom de fonctions, lettres, chaînes de caractères et commentaires présents dans ce code source.

❖ Design recovery
Le but du Design recovery est d'offrir une vue dans un niveau d'abstraction plus élevé que l'artéfact d'entrée du système étudié (que ça soit une implémentation bas niveau comme le code source ou même depuis tout autre artéfact présent dans l'analyse du système).

2.5 Et si il était possible ...

L'idée proposée dans ce mémoire est d'explorer et lister quelles sont les techniques qui permettraient de d'identifier une forme de variabilité. Si il était possible de le faire, alors, il serait possible de comparer 2 modèles de variabilité (un venant de l'analyse et un venant de la rétro-ingénierie) afin d'en vérifier l'exactitude du second, ainsi que de vérifier si l'implémentation de la variabilité correspond bien à celle espérée.
Par exemple, s'il existe un feature modèle d'origine et que l'on parvient à extraire un second FM de l'implémentation représentant exactement l'implémentation, il serait possible d'en déceler les erreurs et d'éviter tout type de défaillance qui peut s'avérer importante dans les systèmes critiques.

Remarque : il ne s'agirait pas de corriger un FM par rapport au système implémenté comme la technique de correction expliquée précédemment (Henard et al.) mais bien de comparer un FM par rapport à l'analyse de ce système implémenté (donc, par rapport à ce qu'il aurait dû être).

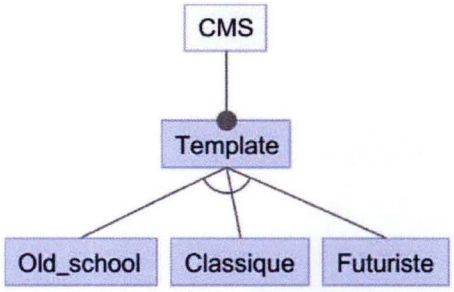


Figure 11 – Erreur dans l'implémentation de variabilité – depuis l'analyse : correct

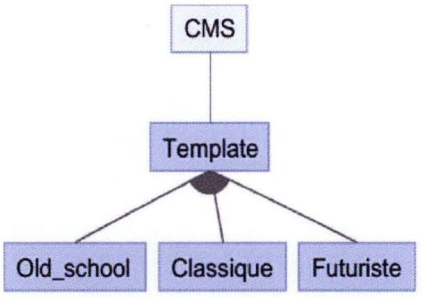


Figure 12 - Erreur dans l'implémentation de variabilité – depuis une technique de rétro-ingénierie : erreur

3 Extraction de la variabilité par le comportement

Lors de la découverte de ce sujet (comprenant son contexte global de recherche) et de multiples lectures à propos des lignes de produits logiciels, la possibilité d'une relation entre le comportement d'un système et la variabilité a germé. Cette relation est basée, sur la représentation connue des transitions systems (TSs) dont les relations peuvent être associées à des features pour produire des feature transitions systems (FTSs).
Pour mieux en connaître le fond, il convient de déterminer ce qu'est un TS.

3.1 Transition systems

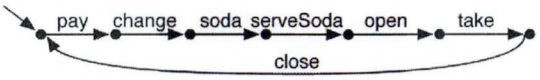
Une définition complète de ce qu'est exactement un TS est proposée par Classen et al. (Classen, et al. 2013) :

> « Un **transition system** est un tuple M = (S, Act, trans, I, AP, L) où :
> - S est un ensemble d'états,
> - Act est un ensemble d'actions,
> - trans ⊆ S x Act x S est une relation (avec (S1, α, S2) ∈ trans, parfois noté : $S1 \xrightarrow{\propto} S2$),
> - I ⊆ S est un ensemble d'états finis,
> - AP est un ensemble de propositions atomiques,
> - L : S → 2^{AP} est une fonction de nommage (qui à chacun des états, associe une proposition atomique devant être satisfaite pour accéder à cet état). »
>
> (traduit par PATINY Mathieu)

> « Une **exécution** (aussi appelé comportement) de M est une séquence finie et non vide $\pi = S_0 \propto_1 S_1 \propto_2$ avec : $S_0 \in I$ tel que $S_i \xrightarrow{\propto_{i+1}} S_{i+1}$ pour tout 0 i.
> La sémantique d'un TS est écrite $[[t]]_{TS}$, est donnée par son ensemble d'exécutions. »
>
> (traduit par PATINY Mathieu)

Un TS est donc représentable sous la forme d'un diagramme d'états dont les transitions sont nommées.
Un exemple repris dans plusieurs publications de l'Université de Namur est celui d'une machine à café.

L'ensemble des exécutions possibles de cette machine à café est repris dans la Figure 13 et Figure 14.



(a) Basic vending machine

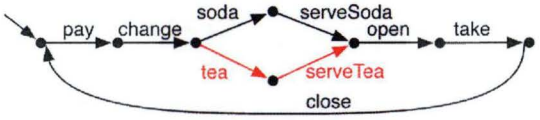
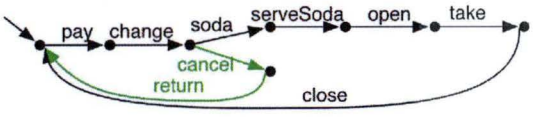
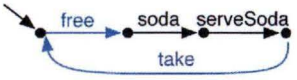


Figure 13 – Exemple « machine à café », exécutions (1) (Classen, et al. 2013)



(c) With a cancel purchase function



(d) Distributing soda for free

Figure 14 - Exemple « machine à café », exécutions (2) (Classen, et al. 2013)

La Figure 15 étant quant à elle, le FD voulu pour cette machine à café.

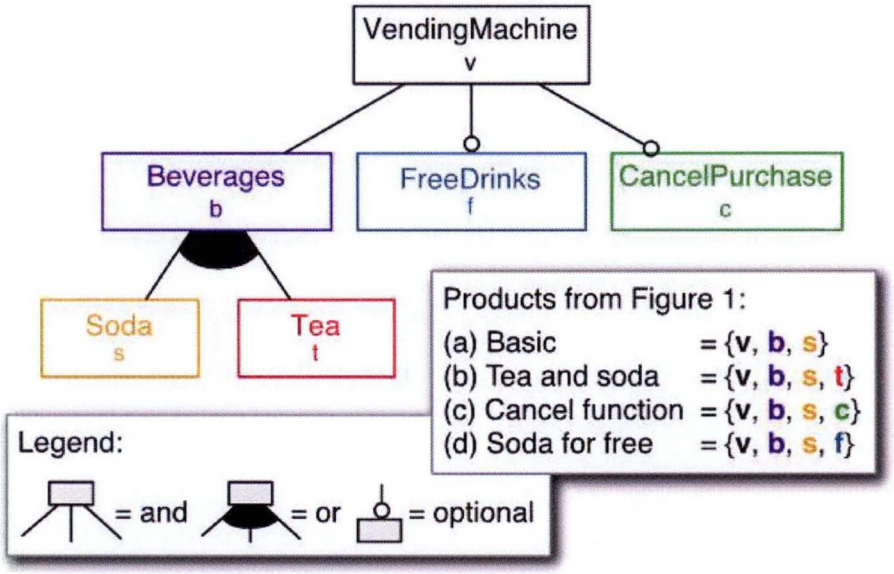


Figure 15 – Exemple « machine à café », feature diagrams (Classen, et al. 2013)

L'optique même des lignes de produits logiciels se basant sur la réutilisabilité des composants, introduit donc la fusion de ces modèles (Figure 13 et Figure 14) dans le but de partager les états et ainsi former une véritable ligne de produits logiciels. Le résultat de type TS correspondant est celui représenté par la Figure 16 ci-après.

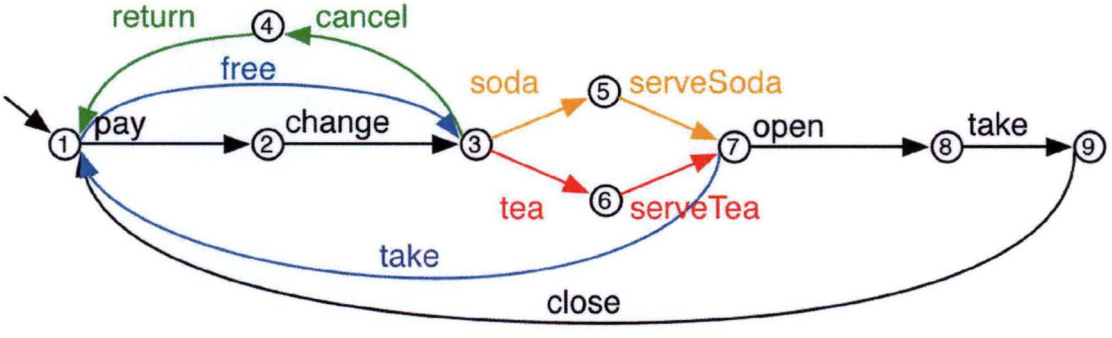


Figure 16 - Exemple « machine à café », fusion des exécutions (Classen, et al. 2013)

De la sorte, un TS permet d'exprimer une suite d'actions basculant le système d'un état à un autre. Cette suite définit le comportement que doit adopter un système.
Ce genre de modèle n'est donc pas adapté au développement en ligne de produits logiciels dans le sens où il ne représente aucune contrainte s'il est donné tel quel. Dans la Figure 15, une série de produits est imposée. Par exemple, le produit « Basic {v, b, s} » ne peut amener le système dans l'état 4 qui requière une fonction d'annulation. Or, la Figure 16 ne mentionne nulle part cette contrainte.
Dès lors, Classen et al. proposent une version améliorée incluant ces contraintes : les Feature Transitions Systems (FTS).

3.2 Feature transitions systems

Classen et al. (Classen, et al. 2013) définissent un feature transition systems comme suit :

> « Un **feature transition system** est un tuple (S, Act, trans, I, AP, L, d, γ),
> - S, Act, trans, I, AP, L comme défini par les FTSs,
> - d est un FD,
> - γ : trans $\rightarrow$ IB(N) est une fonction totale qui nomme chaque transition avec une feature expression, c'est-à-dire une expression booléenne sur la présence de la feature. Un produit p $\subseteq$ P(N) définit une vraie affectation pour les variables dans γ(t). L'ensemble des affectations satisfaisantes, noté [[γ(t)]], est un ensemble de produits. »
>
> (traduit par PATINY Mathieu)

Par rapport aux TSs, les FTSs lient à chaque TS un FD ainsi qu'une fonction exprimant l'autorisation (sous forme de booléen) pour un produit, d'offrir une transition ou non. Le FTS correspondant à l'exemple serait donc la Figure 17 ci-après.

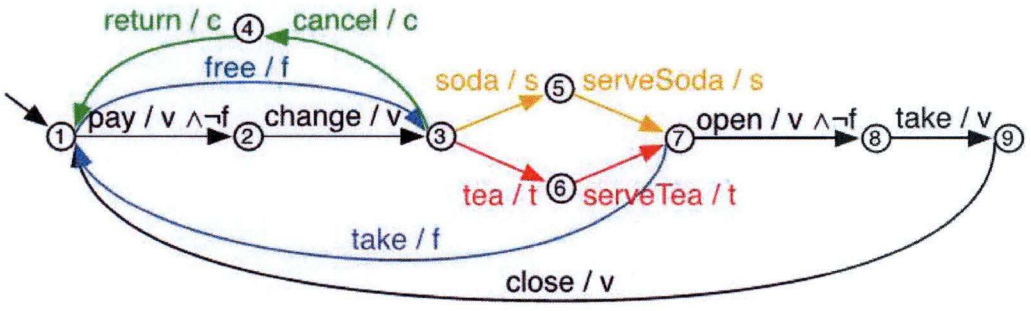


Figure 17 - Exemple « machine à café », feature transitions system (Classen, et al. 2013)

Dès lors, le produit « Basic » ne pourrait plus amener le système dans l'état 4 car la contrainte de présence de la feature c n'est pas respectée.

Cette modélisation ouvre des perspectives multiples quant à la vérification des lignes de produits logiciels ((Classen, et al. 2013), (Devroey, Perrouin et Legay, et al. 2014), (Devroey, Perrouin et Schobbens, 2014), ...) mais aussi de potentielles perspectives vers la rétro-ingénierie de systèmes.

3.3 Vers une rétro-ingénierie de FTS

Une idée explorable serait, depuis un système implémenté en lignes de produits logiciels, d'extraire, pour chaque produit, ses exécutions possibles afin d'en déterminer les TSs correspondants. Il serait ensuite possible de les fusionner tout en labélisant les exécutions : ces labels correspondraient aux features. Il serait ainsi possible d'exprimer la variabilité d'une ligne de produits logiciels depuis l'ensemble de ses implémentations.

Dès lors, il est intéressant de vérifier si la littérature propose des solutions/approches permettant, depuis l'analyse du comportement d'un système, de retourner de l'information donnant une indication sur la variabilité des systèmes.

Objectif du mémoire

Dans le contexte global de recherche qui encadre de mémoire, un des buts est d'extraire des modèles comportementaux prenant en compte la variabilité (typiquement les FTS) de systèmes implémentés. Selon certains chercheurs dans le domaine[7], il n'existe pas encore de méthode permettant d'effectuer cette opération. Il est dès lors intéressant d'effectuer des recherches approfondies d'investigation sur toute technique de rétro-ingénierie permettant l'expression de la variabilité (but principal du mémoire) et ce, en tenant bien compte que certains modèles comportementaux englobent de la variabilité.

Ce type de recherche peut être mené suivant une méthode particulière nommée « systematic mapping study ».

[7] Après de multiples discutions avec les co-promoteurs de ce mémoire.

4 Une « systematic mapping study »

Le domaine de la rétro-ingénierie de la variabilité de système web étant un domaine assez récent, à l'heure actuelle, il existe néanmoins certaines publications scientifiques sur le sujet explorant des pistes différentes. Mais, il n'existe pas encore de référence(s) en la matière permettant d'avoir une vue claire et précise sur ce qui est possible de faire dans ce domaine. C'est pourquoi, ce mémoire consiste à découvrir et classifier ces directions déjà explorées par les chercheurs. Le meilleur moyen d'effectuer ce genre de recherche, est de passer par une technique particulière nommée : « systematic mapping study ». Cette méthode permet l'exploration d'un domaine particulier et la classification systématique de la littérature s'y rapportant. Tout au long du chapitre Chapitre III, le protocole choisi comme préconisé par cette méthode est détaillé et appliqué pas à pas.

4.1 Systematic mapping study VS systematic literature review

Il est important de ne pas confondre « systematic mapping study » avec « systematic literature review ». Bien que ces deux techniques présentent plusieurs similitudes, plusieurs différences fondamentales les distinguent (Petersen, et al. s.d.)[8] :

- Le but
  Le but d'une systematic literature review est essentiellement d'identifier les bonnes pratiques sur base d'études empiriques. A contrario, le but d'une mapping study est de classer des publications en fonction d'une thématique (Il ne s'agit pas d'étude détaillée de publications. Il n'est donc pas possible d'identifier les bonnes pratiques présentes à l'intérieur de celles-ci).

- Le processus
  1. Dans une mapping study, le contenu des publications n'est pas évalué suivant leurs qualités (contrairement à une systematic literature review) car ce n'est pas leur son principal. Ceci dit, une évaluation de la qualité à chaque publication, basée sur le contenu et la formulation au sens large (ex. : le niveau de détail, la complétude ...) est un plus.

  2. Dans une mapping study, la méthode utilisée est une méthode basée sur la thématique des publications. On ne lit que certaines parties des publications dans le but d'être en mesure d'effectuer la classification. Cela contraste donc avec une systematic literature review qui explorera tout le contenu en s'attachant aux détails afin de pouvoir fournir une information plus complète.

- La profondeur
  Dans le cadre d'une mapping study, il est possible de classer beaucoup plus d'articles que dans une systematic literature review. Le but n'étant pas d'évaluer en détail chaque publication, un nombre plus grand de publications peut être dès lors traité.

[8] Une liste exhaustive des différences est disponible dans la publication de Peterson et al. (Petersen, et al. s.d.)

- Accessibilité et pertinence dans l'industrie
  L'article de Peterson et al. mentionne que les industries du logiciel trouvent un intérêt dans ces deux systèmes. Cependant, elles expriment une différence importante dans l'utilité de ces 2 techniques. Dans le premier cas, une mapping study permet de synthétiser et faciliter le transfert de résultats aux praticiens. Quant aux systematic literature reviews, leurs résultats étant plus basés sur une analyse profonde et une validité empirique, elles sont plus utilisées par les praticiens.

Malgré ces différences, ces deux techniques peuvent néanmoins être complémentaires. Une mapping study pourrait, par exemple, être une première étape vers une literature review.
Cependant, ce n'est pas toujours le cas, dans le cadre de ce mémoire, une mapping study est suffisante pour répondre au besoin d'éclaircissement des techniques applicables dans le domaine.

4.2 Outils utilisés dans le cadre de ce mémoire

Ce type de procédé est très coûteux au niveau du temps et peut amener très facilement à des erreurs dues à la grande importance qu'a le meneur de l'étude sur les choix qu'il prend. Par la suite, un protocole (une manière très rigoureuse de travailler suivant un cadre particulier) précis est appliqué dans lequel une grande partie des étapes est menée de manière manuelle. Afin de limiter au strict minimum les erreurs pouvant être commises, il convient de s'entourer d'une sélection judicieuse d'outils facilitant la tâche du meneur afin qu'il ne doive s'occuper uniquement que de la partie « intelligente » (il est entendu par là, tous les choix et tris à effectuer) du protocole (voir ci-après).

Parmi ces outils disponibles, certains sont des références en la matière et fournissent une multitude de fonctionnalités pratiques.

- Gestion de bibliographie
  Dans la mapping study menée au chapitre Chapitre III ci-après, le logiciel (version standalone ou web) « Mendeley »[9] a été utilisé. Il apporte une suite de fonctionnalités utiles telles comme :
  - un *moteur de recherche* intégré (acceptant les requêtes intelligentes de type « 'mot1' and 'mot2' »),
  - un *système de collaboration* permettant de partager des publications entre chercheurs,
  - un *système d'annotation* de commentaire directement dans les publications,
  - une *synchronisation* via le cloud,
  - des *plugins* pour les grandes suites tels que les éditeurs de texte Microsoft Office Word, Mac Word, LibreOffice Writer ... permettant l'insertion automatique de bibliographie,

[9] https://www.mendeley.com/dashboard

- des *plugins* permettant, depuis la visite d'un site Internet d'une librairie en ligne, d'ajouter en 2 clics une référence dans une bibliothèque de la publication affichée,
- l'*exportation* et *importation* de bibliothèques vers d'autres formats/logiciels,
- l'*autocomplétion* (non obligatoire) d'informations (ex. : la liste des auteurs, l'abstract, les mots-clés d'auteurs, l'année de publication, le journal de parution ...) concernant une publication par rapport à sa propre base de connaissance.

L'ensemble de ces fonctionnalités fait de lui un outil pratique et parfait pour une gestion complète des multiples publications récupérées.
Il existe d'autres logiciels comme « Zotero » (https://www.zotero.org) et « JabRef » (http://jabref.sourceforge.net) qui entrent aussi dans cette gamme de logiciels.

❖ Manipulation d'informations

Dans le protocole, certaines étapes demandent des manipulations de dizaines (ou plus) de données. Ces données doivent êtres classifiées et souvent manipulées (scindées, regroupées, comptées ...). A ces fins, les logiciels de type tableur sont très pratiques. Dans cette mapping study, le logiciel « Microsoft Office Excel » a été utilisé mais tout autre tableur peut aussi convenir.

❖ Création de graphiques

Graphiques simples

Les mapping study contiennent plusieurs types de graphiques permettant d'exprimer les résultats. L'ensemble des graphiques illustrés dans le chapitre Chapitre III proviennent tous du logiciel « Microsoft Office Excel ». Ce logiciel a le grand avantage de pouvoir créer une multitude de graphiques en peu de temps et surtout, basés sur des données reprises dans son classeur et sur lesquelles, il est possible d'effectuer de nombreuses manipulations.
Comparé à d'autres logiciels proposant, pour les fonctions basiques, les mêmes services tels que « LibreOffice calc » (https://www.libreoffice.org), « Google Docs Spreadsheet » (https://www.google.com/sheets/about) ou encore « Zoho Sheet » (http://www.zoho.com) il a l'avantage de s'intégrer totalement dans le logiciel « Microsoft Office Word » avec le quel ce mémoire est écrit.

Graphiques à bulles

Pour effectuer des graphiques à bulles, le logiciel « R » (http://www.r-project.org) est utilisé. C'est un logiciel complet et OpenSource orienté statistiques et graphiques. Il permet en outre, de dessiner des graphiques à bulles (via le plugin « ggplot ») avec en abscisse et ordonnée du texte, et ce, en quelques lignes de commandes. Il fournit un environnement de type IDE permettant l'exécution contrôlée (avec gestion d'erreurs) de scripts. Il est un concurrent direct de SAS (Statistical Analysis Software) qui lui, n'est pas OpenSource.

Chapitre III. Mapping study

Définition du protocole

Comme mentionné par Peterson et al. (Petersen, et al. s.d.), une mapping study consiste à créer, depuis une procédure et un sujet de recherche bien déterminés (voir ci-après), un document permettant d'avoir une vue d'ensemble des publications scientifiques existantes sur ce sujet et d'en déduire les lignes directrices pour les futures recherches (ex. : quels sont les domaines grandement ou peu étudiés, les techniques connues, approuvées ou en voie d'exploration ...).

> Une **publication** est tout document écrit visant à transmettre un savoir, une connaissance. Ex. : un article, un rapport de recherche, un mémoire, une thèse, des slides, etc.

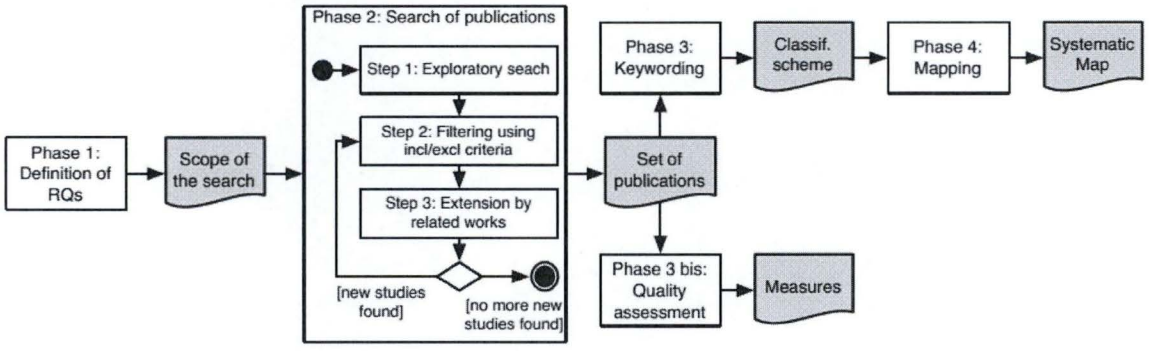


Figure 18 - Systematic mapping process (Devroey, Perrouin, et al., 2014)

La Figure 18 représente un protocole utilisable[10] dans le cadre d'une « mapping study ». Dans le cadre de cette mapping study, c'est ce protocole qui est utilisé. Ce protocole est découpé en quatre grandes phases.

Dans un premier temps, la phase 1 permet de définir la/les question(s) de recherche. Cette étape permet ainsi de déterminer le cadre exact de l'investigation sur le sujet et donc, implicitement, la ligne de conduite de la mapping study (à la fin de cette dernière, il sera possible de répondre aux différentes questions de cette 1ère phase).

Une seconde phase propose une méthode afin d'exfiltrer une collection de publications scientifiques (depuis différentes sources de données) ayant un intérêt pour le domaine de recherche étudié. Un processus bien défini permettra la récupération d'une collection complète et pertinente.

Depuis cette collection, une troisième phase consistera à l'extraction de mots-clés dans un but de classer les différentes publications, mais aussi, de passer via une phase parallèle (phase 3 bis) permettant d'établir une mesure de la qualité de ces publications scientifiques (suivant un ensemble de critères à définir).

Enfin, une 4ème phase générera une « map » sous forme de graphes à bulles. Ce dernier permettra d'avoir une vue rapide et globale suivant différentes facettes. Ces dernières seront définies afin de répondre au mieux à la/aux question(s) de recherche.

[10] Certaines publications proposent des techniques légèrement différentes mais toujours très similaires (basées sur la technique proposée par Petersen (Petersen, et al. s.d.)).

1 Phase 1 – définir les questions de recherche

Dans le but de délimiter le cadre des littératures recherchées et d'en retirer les mots-clés afin d'effectuer des recherches pertinentes, la méthode propose la création de questions de recherche (QRs).

Ainsi, dans le cadre de cette étude, les questions de recherche permettront *d'identifier quelles sont les différentes approches connues applicables (ou non) pour la rétro-ingénierie de la variabilité (RIV) de système web configurable / à haute variabilité (représentables sous forme de lignes de produits logiciels (LPL)).*

Les QRs (triées par ordre d'importance) sont détaillées ci-après.

QR1. Quelles sont les **différentes techniques connues** de RIV ?

QR2. Quelles sont les différentes ***sources d'entrée*** sur lesquelles se basent ces techniques ?

QR3. Quels sont les différents ***artéfacts de sortie*** produits par ces techniques ?

QR4. Quels sont les différents ***mécanismes connus*** permettant la gestion (l'expression et la manipulation) de la variabilité au sein des techniques de RIV?
Sous-question : Quels sont les ***artéfacts particuliers intermédiaires*** (orientés variabilité) utilisés au sein de ces techniques [11] ?

QR5. Existe-t-il des procédés pour ***évaluer la couverture réelle*** d'une technique de rétro-ingénierie appliquée ?

QR6. Quels sont les ***types de publications*** ainsi que leurs ***évolutions dans le temps*** concernant la RIV de LPL ?

QR7. Existe-t-il des ***outils particuliers*** intervenant dans les mécanismes de RIV (exclusivement pour ou non) ?

[11] Il s'agit de tout artéfact (modèle, langage, pattern, formule ...) intermédiaire (orienté variabilité) sur lesquels les techniques de RIV se basent.

2 Phase 2 – exécuter et filtrer les recherches

Dans cette seconde phase, la méthode propose un processus en 3 étapes :

1. Explorer les bases de données en ligne sur fond de chaînes de caractères de recherches.
2. Filtrer les résultats obtenus via des critères d'inclusion / exclusion.
3. Sur base des résultats de l'étape 2, récupérer les références citées (sur base de pré-critères[12]) dans ces documents et les filtrer via l'étape 2.

Auparavant, il est nécessaire de récupérer une liste de sources de données où l'information peut potentiellement se trouver ainsi que des mots-clés qui, assemblés via des opérateurs de type booléens (typiquement OR (|) et AND (&)) formeront les contraintes de la recherche.

❖ Sources de données

➢ Librairies en ligne

Les bases de données en ligne considérées sont les suivantes :

Google scholar (https://scholar.google.be)	Microsoft Academic Search (http://academic.research.microsoft.com)	dblp computer science bibliography (http://dblp.uni-trier.de)	ScienceDirect (http://www.sciencedirect.com)
ACM DL DIGITAL LIBRARY ACM – Digital Library (http://dl.acm.org)	IET Digital Library The Institution of Engineering and Technology (http://digital-library.theiet.org/)	The Collection of Computer Science Bibliographies (http://liinwww.ira.uka.de/bibliography)	Open Grey System for Information on Grey Literature in Europe (http://www.opengrey.eu)
DOAJ DIRECTORY OF OPEN ACCESS JOURNALS Directory of open access journals (http://doaj.org)	OpenDOAR The Directory of Open Access Repositories (HTTP://WWW.OPENDOAR.ORG)	ccsd Centre pour la Communication Scientifique Directe (http://www.ccsd.cnrs.fr)	RefSeer RefSeer: A Citation Recommendation System ~~(http://refseer.ist.psu.edu)~~
DSpace@MIT MIT Open Access Articles (https://dspace.mit.edu)	IOWA STATE UNIVERSITY ISU Computer Science Archives (http://archives.cs.iastate.edu)	Cornell University Library Cornell Universiry Library (http://arxiv.org)	CiteSeer$^{x}_{\beta}$ CiteSeerX (http://citeseerx.ist.psu.edu)

[12] Ex. : référence vers des documents visant le sujet déterminé dans la phase 1.

Electronic Theses and Dissertations website of the Katholieke Hogeschool Kempen (http://doks2.khk.be)	Repositorio institucional de la UNLP (http://sedici.unlp.edu.ar)	FreeSearch Toward computer science ideas (http://dblp.kbs.uni-hannover.de)	L'IEEE Computer Society (http://www.computer.org)
IEEE Xplore Digital Library (http://ieeexplore.ieee.org)			

Tableau 4 – Librairies en lignes

- **Littérature grise**

  Telle que définie par Schöpfel (Schöpfel 2011), la littérature grise est tout document qui n'est pas publié via les canaux habituels et en dehors des circuits commerciaux de l'édition et diffusions. Ces documents sont donc plus difficiles d'accès (typiquement : rapports d'études ou de recherches, thèses ...).

  Pour en avoir connaissance, une technique consiste à analyser un document et extraire les documents référencés qui ont le même centre d'intérêt que le sujet de recherche (technique appliquée dans l'étape 3).

- **Mendeley**

  Le logiciel « Mendeley » est (entre autres) un référenceur académique gratuit (Mendeley s.d.). Il permet d'effectuer une recherche dans une multitude de « repository » de publications en tout genre. Ainsi, il peut aussi se classer dans les sources de données utilisables.

❖ Mots-clés

Dans un premier temps, une liste des mots-clés permettant de répondre aux questions de recherche est créée[13] : *reverse engineering, reverse-engineering, reversing, reverse architecting, reverse behavior, reverse from, get behavior, extraction, program comprehension.*

Chaque recherche doit impérativement concerner les LPL (systèmes à haute variabilité (SHV) ou plugin-based). De ce fait, il convient donc d'ajouter au moins un des mots-clés suivants aux recherches : *product line, product family, feature-based, system family, software reusability, component-based, reuse context, reuse oriented, feature-oriented, variability intensive system, configurable system, plugin-based, feature models, SPL.*

Les mots-clés sont ciblés mais assez larges pour, par la suite, avoir la possibilité de récupérer un maximum de publications et de n'en laisser aucune de côté du fait qu'elles pourraient utiliser un mot synonyme ou particulier non standard. Les publications sortant du cadre étant supprimées par la suite, cela alourdit le travail fourni par le meneur de l'étude mais permet d'avoir une étude plus exhaustive.

Dans le cadre de l'orientation du sujet de ce mémoire, il est intéressant de regarder jusqu'à quel point il est possible d'affiner les recherches sur le sujet qui est, d'après des chercheurs dans le domaine[14], assez récent et pour l'heure, pas largement approfondi. En conséquence, sur chaque source de données, les mots-clés « *web* » et « *internet* » sont ajoutés à la chaîne (partie verte dans la chaîne de recherche) afin de tester si oui ou non leur utilisation permet la récupération (par rapport à leurs non-utilisations) de résultats complets, cohérents et surtout précis (permettant aussi un allègement de l'étape 2) et ce, sans perte de publications pertinentes.

Dans un second temps, en agençant les mots-clés et les opérateurs booléens, un prototype d'une chaîne de recherches (les contraintes d'une recherche) est généré :

("reverse engineering" OR "reverse-engineering" OR "reverse architecting" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension")
AND
("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
AND
("web" OR "internet")

Cependant, certaines sources de données n'étant pas uniquement basées sur la science de l'informatique, il peut être nécessaire d'ajouter un mot-clé afin de préciser le domaine[15] (ou le préciser via le GUI / champs de recherche de la ressource) : "software engineering".

[13] Cette liste a été réalisée sur base d'un ensemble cohérent lectures sur le sujet, mais aussi sur base des conseils d'experts du domaine.
[14] Il est question des co-promoteurs du mémoire.
[15] La médecine et la biologie sont des secteurs dont les mots-clés sont de temps en temps producteurs de résultats non désirés.

Remarque : lors de la détermination des différents mots-clés précédents, le langage « Anglais » est utilisé. Ce choix est basé sur de grands constats. Le premier est que l'anglais est la langue universelle utilisée dans la littérature de type scientifique (Romdhane 1995-1996). Le second est que, dans le milieu de l'informatique, la langue anglaise est aussi la langue universelle. Étant donné le but de la mapping study, il est donc naturel d'utiliser ce langage grâce auquel un maximum de documents pourra être récupéré. Cela implique, potentiellement, que certains documents (tels que les littératures grises non anglophones) ne pourront être découverts.

2.1 Effectuer les recherches

Durant cette première étape, la recherche des publications à travers les différentes sources de données est menée. Pour ce faire, sur chaque source de données, une recherche est effectuée avec la contrainte décrite dans la chaîne de recherche précédente (les recherches détaillées sont disponibles en annexe III). Un premier ensemble « brut » de publications est ainsi récupéré.

Lors de l'exécution des recherches, le choix de l'endroit du texte recherché est, parfois, spécifiable. Dans cette exécution, les recherches se font, au maximum dans l'*abstract*, les *mots-clés* (de l'auteur ou générés par la source de données) et le *titre* de la publication. Ceci dit, certaines sources ne permettent pas une recherche dans ces 3 éléments. Dans ce cas, une recherche « par défaut » est effectuée ou, si possible, uniquement dans le titre si le nombre de résultats « par défaut » est trop important.

Remarque : il est à remarquer que certaines sources de données utilisent des moyens lexicaux et syntaxiques différents d'exprimer la chaîne de recherche (certaines ne sont pas toujours assez sophistiquées pour effectuer des recherches longues et/ou plus ou moins complexes en une seule requête[16]). La recherche peut donc se trouver divisée en x parts. Une comparaison afin de supprimer les littératures communes peut alors être nécessaire. Il convient de comprendre ces différents lexiques et syntaxes ainsi que d'adapter la chaîne de recherche à ces derniers. Par exemple, une recherche portant sur l'expression « reverse engineering » dans le titre des littératures se traduit par :

- « ti:"reverse engineering" » sur la ressource en ligne « The Collection of Computer Science Bibliographies »
- « intitle:"reverse engineering" » dans la ressource « Google scholar »
- « "Document Title":"reverse engineering" » dans la ressource « IEEE Xplore Digital Library »
- « Title:"reverse engineering" » (sensible à la casse) dans la ressource « ACM – Digital Library »

La Figure 19 ci-après représente le graphique reprenant, pour chaque source, le nombre de littératures trouvé via la chaîne de recherche précédemment décrite[17] avec et sans les mots-clés « web » et « Internet ». Ces données sont dites « brutes » car aucun tri ni sélection n'a encore été effectué.

---

[16] Ex. 1 : La ressource en ligne « Centre pour la Communication Scientifique Directe - HAL archives-ouvertes.fr » ne permet pas plus de 300 caractères dans son champs de recherche.
Ex. 2 : La ressource « IEEE Xplore Digital Library » n'autorise les recherches d'au maximum 15 termes

[17] Ces résultats sont basés sur des recherches ayant été effectuées entre le 02/02/2015 et le 15/03/2015. En dehors de ces dates, les résultats peuvent se trouver modifiés.

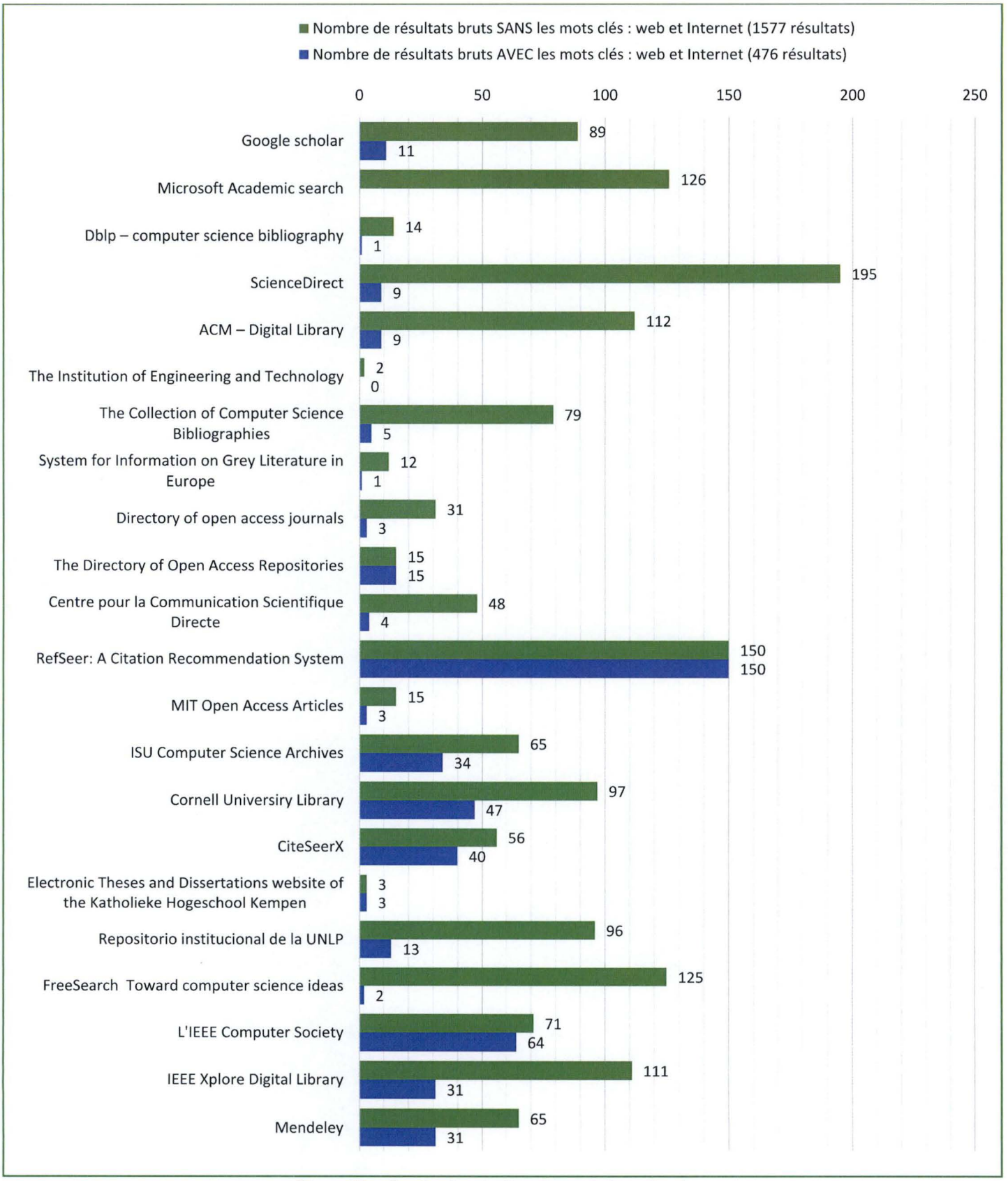


Figure 19 – Graphe des résultats bruts de la recherche

Remarque : dans certains cas, il est peut-être nécessaire de raccourcir la chaîne de recherche afin de trouver des résultats. En effet, certains moteurs de recherche ne retournent aucun résultat avec la chaîne précédemment construite, mais propose des résultats pertinents avec la chaîne suivante (construite par simplification des mots-clés) :

("reverse" OR "get behavior" OR "extraction" OR "program comprehension") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL")

Ces résultats étant bruts, un processus préliminaire d'épuration des résultats (avant d'entamer la phase 2) permet l'élimination des résultats « hors sujet ». Par là, il est entendu tout résultat ne se trouvant pas dans le domaine de l'informatique et de l'ingénierie logicielle (ex. : sciences de la physique, résultats orientés médecine neuronale et reconnaissance facial ...) ou encore n'ayant explicitement pas comme sujet principal le sujet du présent mémoire.

Par la même occasion, il est sans nul doute utile de supprimer les doublons au sein même d'une source de données (certains moteurs de recherches proposent plusieurs fois la même publication à cause de multiples référencements ou encore suivant la chaîne de recherche quand celle-ci a dû être découpée pour des raisons techniques. Ex. : Google Scholar, Directory of open access journals, Centre pour la Communication Scientifique Directe...).

2.2 Filtrage des documents

Un filtrage est appliqué sur chaque publication issue de l'étape 1 afin de supprimer les documents non pertinents (ou non voulus) pour le sujet (ne permettant pas de répondre aux QRs). Ce filtrage se fait sur base de critères *d'inclusion* et *d'exclusion*.

Si ces critères se trouvent allégés, il est fort probable que certaines publications soient retenues et influencent négativement le classement final et les réponses aux questions de recherches.

2.2.1 Critères d'inclusion

- Le titre, l'abstract, l'introduction ou la conclusion mentionne explicitement l'idée de la RIV dans un contexte de LPL.
- Le titre, l'abstract, l'introduction ou la conclusion mentionne l'idée de l'adaptation de technique de RIV dans un contexte de LPL.
- La technique de rétro-ingénierie produit un type de modèle exprimant l'idée de la variabilité.
- Si plusieurs publications forment un continuum (ex. : exploration d'une idée dans une publication et modifications de celle-ci dans un second), chaque publication sera gardée car elle serait potentiellement porteuses d'évaluation de techniques et d'évolution de celles-ci.
- Toute publication mentionnant un outil permettant la RIV avec une optique LPL.
- Tout document académique, de recherche ou d'organisation publié ou non publié (voir littérature grise).
- Le titre, l'abstract, l'introduction ou la conclusion mentionnent explicitement l'idée de la RIV dans un contexte de LPL de système web.

2.2.2 Critères d'exclusion

- Tout document ayant une forme de type présentation[18] ou toute autre forme similaire.
- Tout duplicata (ex. : mêmes publications extraites de 2 sources de données différentes).
- Publication incluse dans une autre (toute publication ayant une version détaillée dans une autre littérature[19]).
- Toute littérature non « évaluée par les pairs »[20] s'il existe une version explicitement mentionnée comme « évaluée par les pairs » disponible.
- Tout document où la RIV est uniquement citée à un endroit sans approfondissement.
- Toute publication n'étant pas orientée système de type web.

---

[18] Ex. : PowerPoint, Beamer, Impress, PowToon, Keynote ...

[19] Ex. : La version journal serait gardée mais sa potentielle version « papier de conférence » serait exclue.

[20] Un document « évalué par les pairs »désigne un document ayant été jugé et validé de manière critique par d'autres chercheurs/experts du domaine acceptant ainsi sa qualité.

Dans cette mapping study, 2 recherches sont menées. L'une comprenant les 2 mots-clés « web » et « Internet » et l'autre sans ces derniers.

Les chiffres repris dans la Figure 19 laissent penser qu'un certain nombre de publications existent sur le sujet avec ces 2 mots-clés. Malgré ces chiffres, après avoir appliqué les critères précédents (avec les critères en vert), seules 10 publications (listées en annexe IV) se rapprochent réellement du sujet dont 8 ne rentrent pas totalement dans le sujet et sont donc abandonnées par les critères de filtration précédents.
Seules 2 publications entrent dans le sujet :

[b] S. Marciuska, C. Gencel, and P. Abrahamsson, "Automated feature identification in web applications," *Lecture Notes in Business Information Processing*, vol. 166 LNBIP, pp. 100–114, Nov. 2014.

[i] E. K. Abbasi, M. Acher, P. Heymans, and A. Cleve, "*Reverse engineering web configurators*", 17th European Conference on Software Maintenance and Reengineering (CSMR), IEEE , 2014.

Dès lors, il convient d'en conclure que ce domaine est vraiment peu exploré et surtout que, les publications sur le sujet sont très récentes (moins d'un an).
C'est une conclusion qui n'est pas vraiment étonnante et avait été clairement mentionnée par les chercheurs du domaine (voir ci-avant). Néanmoins, il était utile et nécessaire de faire le travail de recherche afin de valider cette hypothèse.

Dès lors, il faut considérer ce sujet comme « domaine à découvrir » ou « recherche à approfondir avec d'autres critères/contraintes ».

Il est dès lors judicieux **d'élargir le sujet de recherche** en partant d'une recherche sans ces mots-clés[21] et d'élargir ainsi le champ de recherche à tout système informatique.

Après avoir passé les résultats de la phase 1 aux critères de sélection/exclusion, il en ressort 59 résultats (listés en annexe V) correspondant au sujet élargi. L'étape suivante de cette mapping study sera donc basée sur ces résultats. Cette grande différence entre les chiffres bruts (voir Figure 19) et résultats réels est due, majoritairement, à la présence de publications ayant certains mots-clés en commun avec le sujet mais dont le but principal est hors sujet (ex. : « extraction AND feature-based » forment une association possible via la chaîne de recherche mais retournent plusieurs résultats à propos de la détection d'un visage humain)

Remarque : à titre purement informatif, 27 publications sont aussi apparues sans rentrer totalement dans le sujet et 7 autres sont aussi ressorties mais hors du cadre des LPL / SHV.

[21] Le sujet de la mapping study s'élargit donc. Toute autre publication relatant un système autre qu'un système web sera donc prise en compte.

2.3 Recherche approfondie

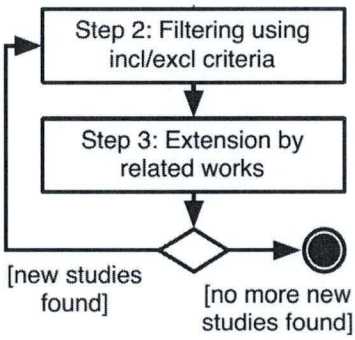


Figure 20 – Step 2 and 3 of Phase 2 de processus de la systematic mapping process (Devroey, Perrouin, et al., 2014)

Depuis les publications précédentes (jugées comme pertinentes), un processus de bouclage est effectué afin de maximiser le nombre de publications récupérables sur le sujet.

Pour toute publication, en parcourant les publications référencées[22] dans les parties suivantes[23], ces références seront listées et filtrées (via l'étape « 2.2 Filtrage des documents ») à leur tour jusqu'à ce qu'il n'y ait plus de documents trouvable et valide par le biais de ce mécanisme :

- l'introduction
- l'overview
- le background
- les travaux en relation
- la conclusion

Les résultats de la 1ère itération sont repris dans Figure 21 ci-dessous. Ces résultats reprennent d'une part, *le nombre total de références dans la publication* ainsi que le *nombre total de références (sans aucun tri) comprises dans les différentes parties* reprises ci-dessus. 1169 références sont récupérées de ces parties (65 % des 1797 que contiennent au total les publications).

Au vu du nombre important de résultats, un premier tri très « laxiste » sur le titre (le titre d'une publication n'étant pas toujours totalement fidèle au contenu de celle-ci) permet de ne récupérer qu'uniquement les *références en relation (de près ou de loin) avec le sujet* de cette mapping study. Des 1169 références, 252 sont récupérées pour la suite.

---

[22] Uniquement celles en anglais ou français. Certains documents comme (C. Luna and A. Gonzalez, "Behavior specification of product lines via feature models and UML statecharts with variabilities," in Proceedings - International Conference of the Chilean Computer Science Society, SCCC, 2008, pp. 32–41.) sont dans une autre langue.

[23] Dans le cas où les références reprises dans ces différentes parties englobent presque la totalité des références de la publication, l'ensemble des références seront analysées afin de ne pas perdre de temps à les récupérer.
Il en va de même pour les publications où le temps de lister toutes les références dans ces parties est plus important que d'analyser directement les références.

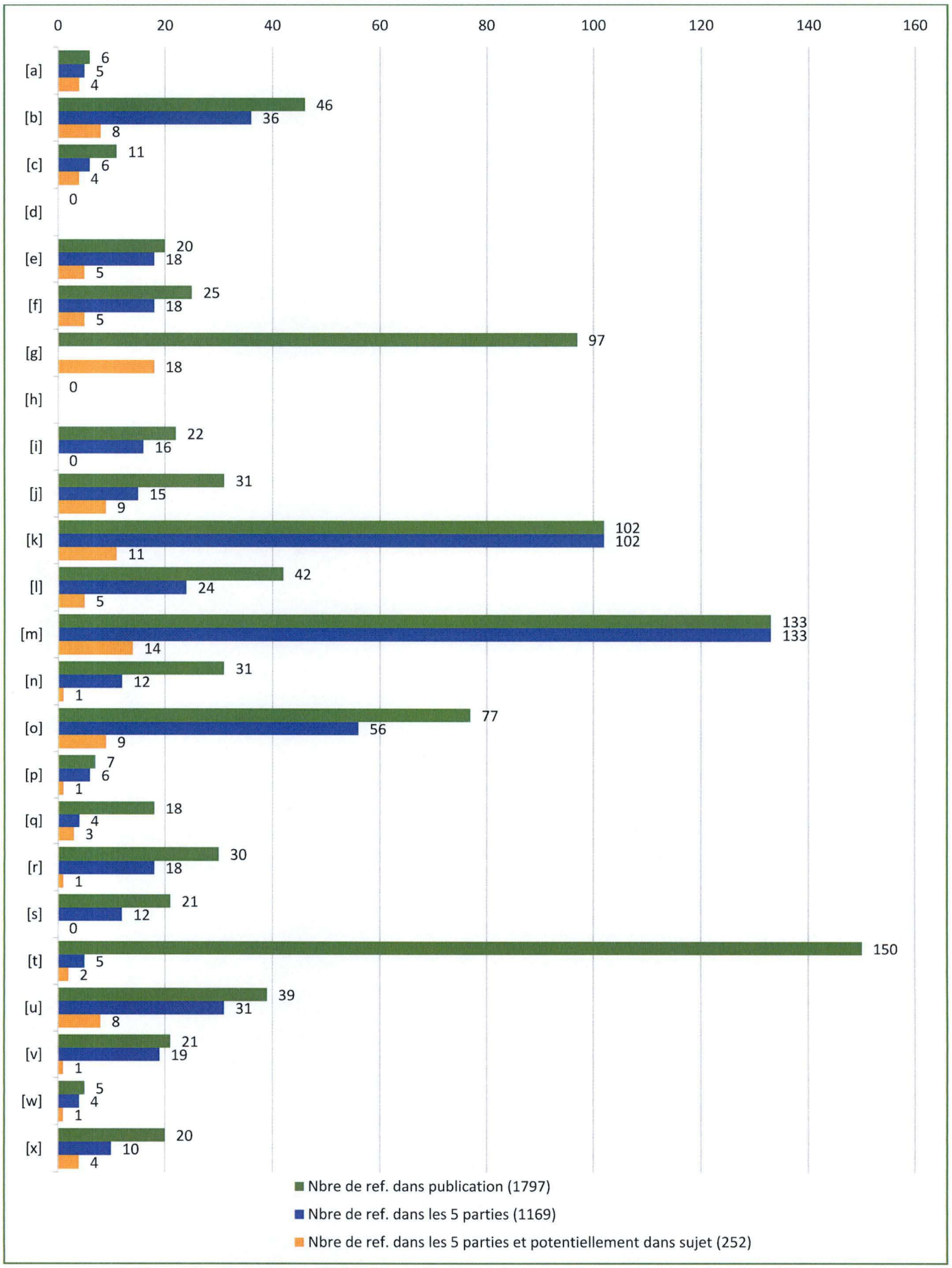


Figure 21 - Recherche approfondie : quelques nombres de références de la 1ère itération (a)

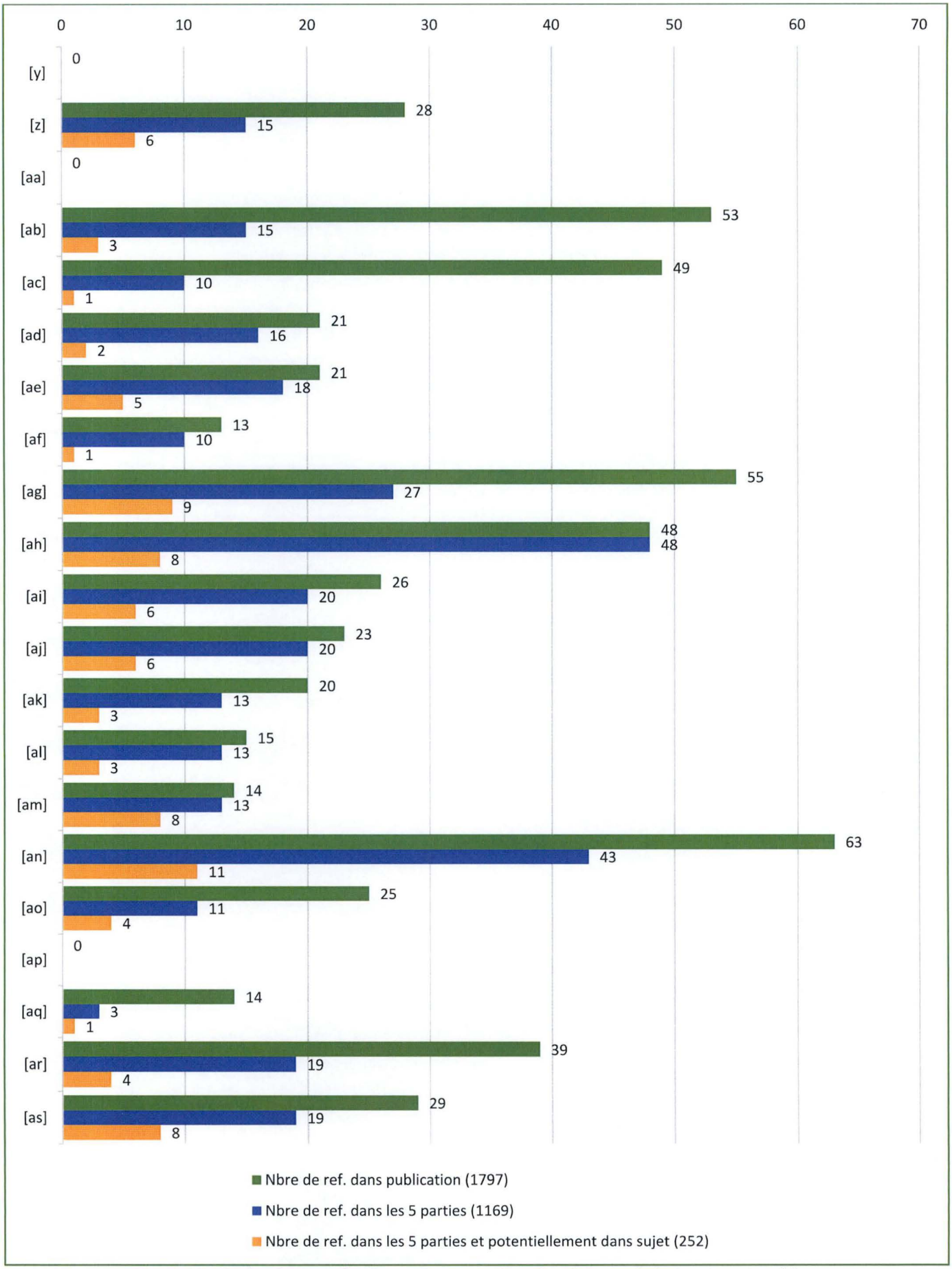


Figure 22 - Recherche approfondie : quelques nombres de références de la 1ère itération (b)

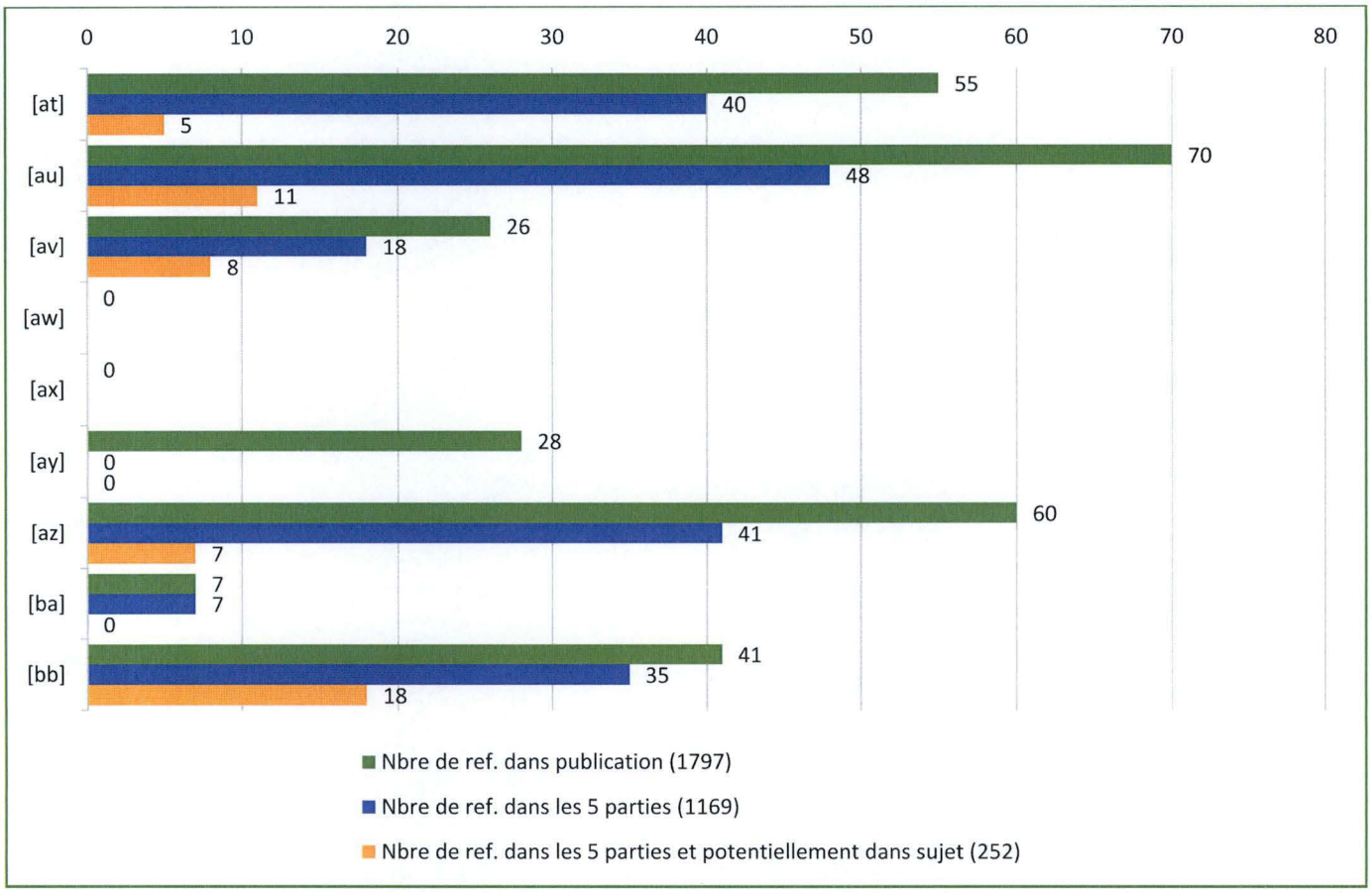


Figure 23 - Recherche approfondie : quelques nombres de références de la 1ère itération (c)

Remarques :

- lors de la récupération des références, aucune n'a été écartée du fait d'être complètement « hors sujet ». Cet indicateur appuie le fait d'avoir des publications pertinentes. Les seules références écartées (avant step 2 de la Figure 20) l'ont été car le titre était assez explicite pour déterminer qu'aucun reverse engineering n'était le sujet principal et/ou non comportemental.
- au sein même des références, plusieurs versions d'une publication peuvent se retrouver. Seule la plus récente est reprise (anticipation des critères d'exclusion) afin de ne pas multiplier le travail à effectuer sachant pertinemment bien que les anciennes versions ne seront pas retenues.

Parmi les références ayant une relation avec le sujet, un certain nombre pointent vers des publications faisant déjà partie de la liste des publications récupérée.
Afin d'éviter du travail inutile, ces publications seront automatiquement retirées de la liste des références.

De plus, certaines références sont des doublons entre elles. Seule la 1ère référence est gardée. Les autres sont marquées comme « doublons entre références».

Au terme de la recherche de la 1ère itération, 175 références sont retenues.

Ci-après, se trouve le graphique reprenant au terme de la procédure de recherche, pour chaque publication, le nombre de publications retenues comme input de l'étape 2 de la recherche approfondie.

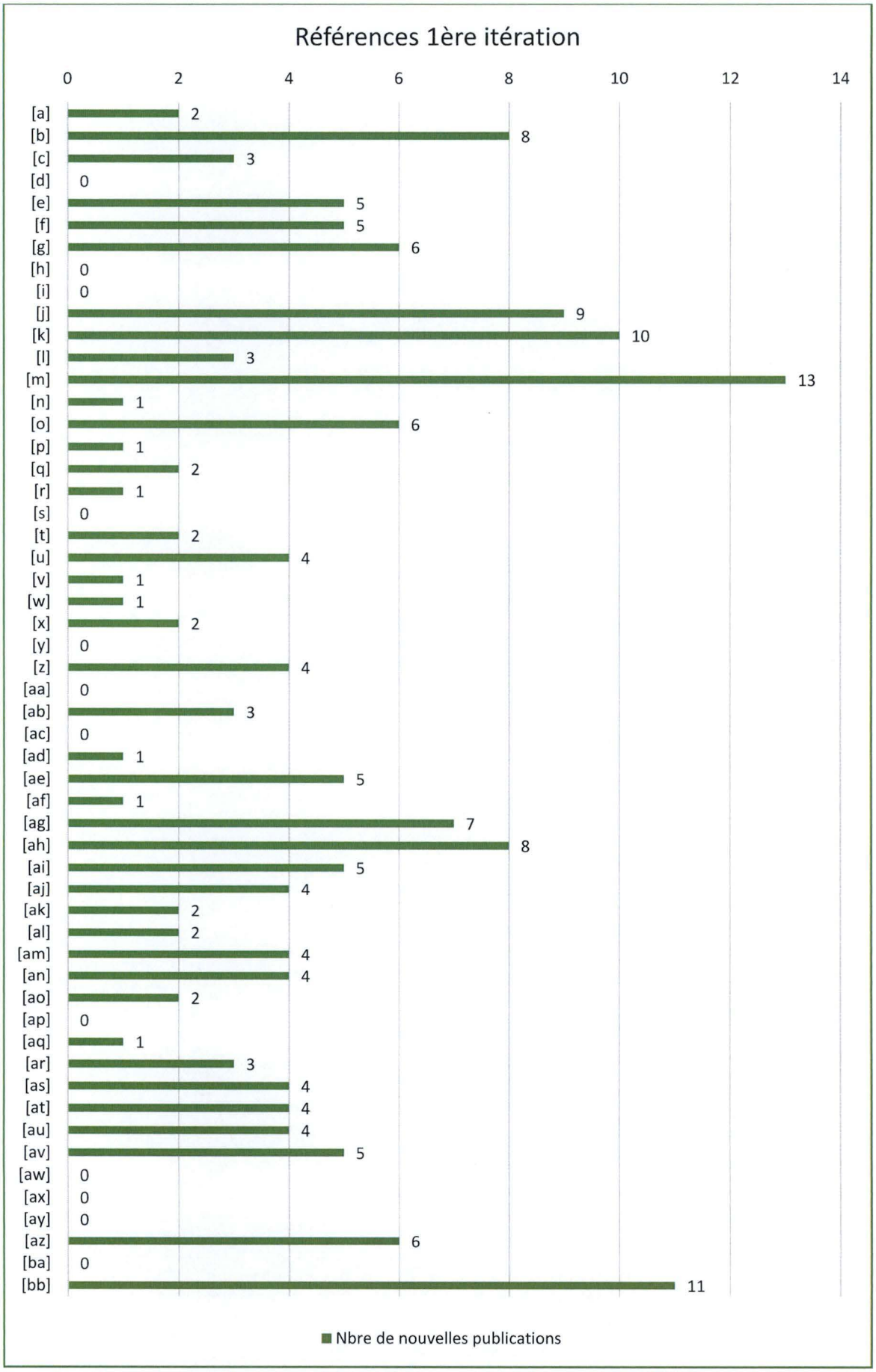


Figure 24 – Recherche approfondie : nombres de références potentiellement à retenir de la 1ère itération

Après application de l'étape 2, seules 15 nouvelles publications sont gardées et ajoutées à la liste des publications retenues.

Le protocole indique une recherche de profondeur infinie. Il convient donc de refaire le même processus pour une 2ème itération. Il n'est présenté ci-après qu'exclusivement la 2ème itération par rapport aux publications récupérées précédemment.

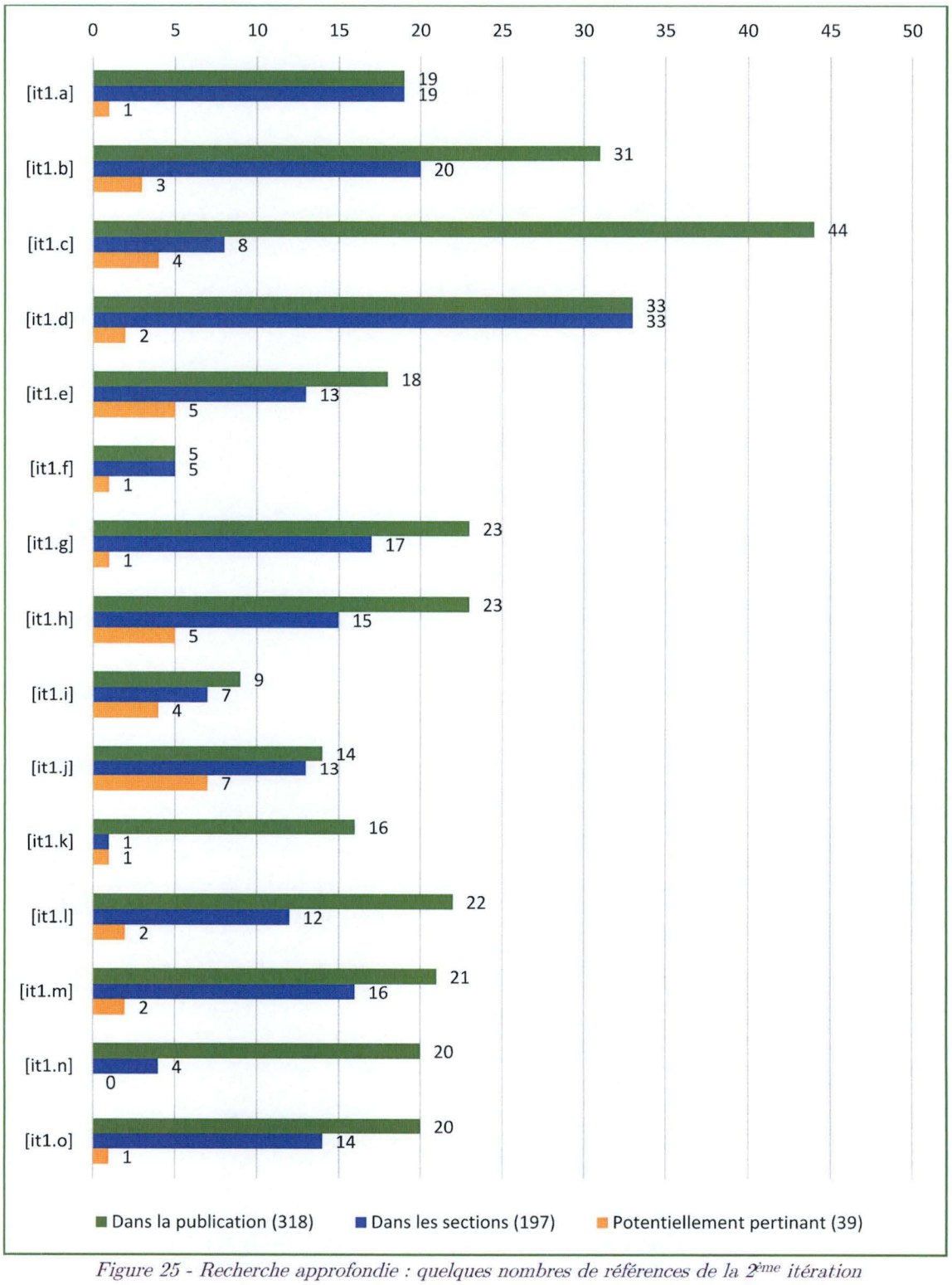


Figure 25 - Recherche approfondie : quelques nombres de références de la 2ème itération

A la vue du petit nombre de références, il n'est pas nécessaire d'effectuer un éventuel pré tri pour alléger l'étape 2.
Après avoir appliqué le filtre de cette dernière étape, 3 nouvelles publications sont gardées et ajoutées.

Une 3ème itération est donc requise.

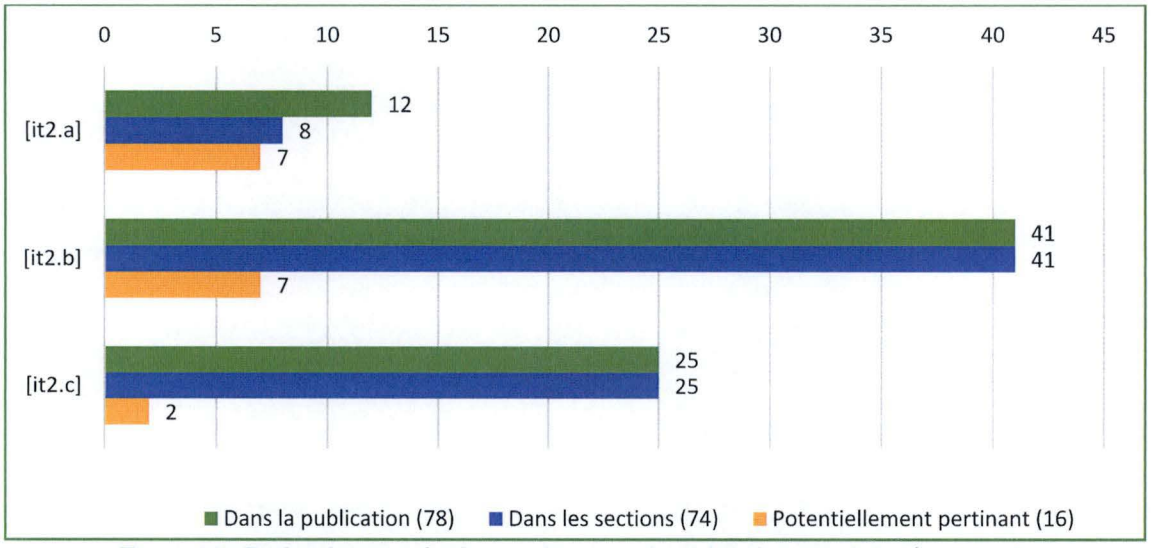


Figure 26 - Recherche approfondie : quelques nombres de références de la 3ème itération

Après avoir filtré les publications référencées, aucune nouvelle publication n'est gardée. En effet, toutes les publications référencées dans cette publication ont déjà été précédemment récupérées.

Remarque : il est normal d'avoir une « petite profondeur » (un nombre de nouvelles références très vite décroissant) étant donné que le sujet est assez récent et que la profondeur donne aussi une indication sur « l'âge » des publications (plus il y a de la profondeur, plus il existe de publications antérieures (anciennes)) mais aussi sur l'étendue des recherches du sujet. Un sujet jeune et peu exploré comme celui-ci ne peut donc avoir une grande quantité de publications totalement pertinentes.

Les étapes de recherche de publications étant terminées, les publications suivantes seront retenues (triées par ordre alphabétique sur le titre) :

[1] N. Weston, R. Chitchyan, and A. Rashid, **"A framework for constructing semantically composable feature models from natural language requirements,"** pp. 211–220, Aug. 2009.

[2] R. Al-Msie'Deen, A. D. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, **"A methodology to recover feature models from object-oriented source code,"** VARY'2012: VARiability for You. 30-Oct-2012.

[3] C. Dietrich, R. Tartler, W. Schröder-Preikschat, and D. Lohmann, **"A robust approach for variability extraction from the linux build system,"** in ACM International Conference Proceeding Series, 2012, vol. 1, pp. 21–30.

[4] M. T. Valente, V. Borges, and L. Passos, **"A semi-automatic approach for extracting software product lines,"** IEEE Trans. Softw. Eng., vol. 38, no. 4, pp. 737–754, Jul. 2012.

[5] J. Rubin and M. Chechik, **"A Survey of Feature Location Techniques,"** Domain Engineering, 2013. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-36654-3_2. [Accessed: 27-Apr-2015].

[6] R. Al-msie, a D. Seriai, M. Huchard, and C. Urtado, **"An approach to recover feature models from object-oriented source code,"** Journée Lignes de Produits. pp. 1–12, 2012.

[7] R. E. Lopez-Herrejon, L. Linsbauer, J. a. Galindo, J. a. Parejo, D. Benavides, S. Segura, and A. Egyed, **"An assessment of search-based techniques for reverse engineering feature models,"** J. Syst. Softw., vol. 103, pp. 353–369, May 2014.

[8] C. Luna and A. Gonzalez, **"Behavior specification of product lines via feature models and UML statecharts with variabilities,"** in Proceedings - International Conference of the Chilean Computer Science Society, SCCC, 2008, pp. 32–41.

[9] T. Ziadi, T. Ziadi, L. Hélouët, L. Hélouët, J.-M. Jézéquel, and J.-M. Jézéquel, **"Behaviors Generation From Product Lines Requirements,"** 7th International Conference on UML Modeling Languages and Applications, 2004.

[10] G. Bécan, M. Acher, B. Baudry, and S. Ben Nasr, **"Breathing ontological knowledge into feature model synthesis: an empirical study,"** Empir. Softw. Eng., p. 51, Mar. 2015.

[11] B. Zhang and M. Becker, **"Code-based variability model extraction for software product line improvement,"** in ACM International Conference Proceeding Series, 2012, vol. 2, pp. 91–98.

[12] A. E. El Hamdouni, a. D. Seriai, and M. Huchard, **"Component-based architecture recovery from object oriented systems via relational concept analysis,"** CEUR Workshop Proceedings, 2010. [Online]. Available: http://ceur-ws.org/Vol-672/paper23.pdf. [Accessed: 23-Mar-2015].

[13] O. Greevy and S. Ducasse, **"Correlating features and code using a compact two-sided trace analysis approach,"** in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2005, pp. 314–323.

[14] A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, and G. Saval, **"Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis,"** in 15th IEEE International Requirements Engineering Conference (RE 2007), 2007, pp. 243–253.

[15] J. Feigenspan, C. Kästner, S. Apel, J. Liebig, M. Schulze, R. Dachselt, M. Papendieck, T. Leich, and G. Saake, **"Do background colors improve program comprehension in the #ifdef hell?,"** Empir. Softw. Eng., vol. 18, no. 4, pp. 699–745, May 2013.

[16] S. She, U. Ryssel, N. Andersen, A. Wąsowski, and K. Czarnecki, **"Efficient synthesis of feature models,"** in Information and Software Technology, 2014, vol. 56, no. 9, pp. 1122–1143.

[17] O. Greevy, **"Enriching Reverse Engineering with Feature Analysis,"** Thesis, 2007. [Online]. Available: http://scg.unibe.ch/archive/phd/greevy-phd.pdf. [Accessed: 23-Mar-2015].

[18] M. V. Couto, M. T. Valente, and E. Figueiredo, **"Extracting Software Product Lines: A Case Study Using Conditional Compilation,"** 2011 15th European Conference on Software Maintenance and Reengineering, 2011. [Online]. Available: http://academic.research.microsoft.com/Publication/39260876/extracting-software-product-lines-a-case-study-using-conditional-compilation. [Accessed: 23-Mar-2015].

[19] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire, **"Extraction and evolution of architectural variability models in plugin-based systems,"** Softw. Syst. Model., vol. 13, no. 4, pp. 1–28, Jul. 2013.

[20] B. Zhang, **"Extraction and improvement of conditionally compiled product line code,"** in IEEE International Conference on Program Comprehension, 2012, pp. 257–258.

[21] U. Ryssel, J. Ploennigs, and K. Kabitzsch, **"Extraction of feature models from formal contexts,"** in Proceedings of the 15th International Software Product Line Conference on - SPLC '11, 2011, p. 1.

[22] S. Apel and D. Beyer, **"Feature cohesion in software product lines: an exploratory study,"** in 2011 33rd International Conference on Software Engineering (ICSE), 2011, pp. 421–430.

[23] K. Czarnecki and A. Wasowski, **"Feature diagrams and logics: There and back again,"** Proc. - 11th Int. Softw. Prod. Line Conf. SPLC 2007, pp. 23–32, Sep. 2007.

[24] T. Ziadi, L. Frias, M. A. A. Da Silva, and M. Ziane, **"Feature identification from the source code of product variants,"** in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2012, pp. 417–422.

[25] R. AL-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, **"Feature Location in a Collection of Software Product Variants Using Formal Concept Analysis"**, vol. 7925. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[26] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, **"Feature location in source code: A taxonomy and survey,"** Journal of software: Evolution and Process, 2013. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.380.6285&rep=rep1&type=pdf. [Accessed: 04-Apr-2015].

[27] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, **"Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace,"** in Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, 2007, pp. 234–243.

[28] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans, **"Feature Model Extraction from Large Collections of Informal Product Descriptions,"** in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 290–300.

[29] R. Stoiber and M. Glinz, **"Feature unweaving: Efficient variability extraction and specification for emerging software product lines,"** in 2010 4th International Workshop on Software Product Management, IWSPM 2010, 2010, pp. 53–62.

[30] J. Feigenspan, M. Papendieck, C. Kästner, M. Frisch, and R. Dachselt, **"FeatureCommander: colorful \#ifdef world,"** Proceedings of the 15th International Software Product Line Conference, Volume 2, 2011. [Online]. Available: http://doi.acm.org/10.1145/2019136.2019192. [Accessed: 27-Apr-2015].

[31] V. Alves, F. Calheiros, V. Nepomuceno, A. Menezes, S. Soares, and P. Borba, **"FLiP: Managing software product line extraction and reaction with aspects,"** in Proceedings - 12th International Software Product Line Conference, SPLC 2008, 2008, p. 354.

[32] N. Sannier, M. Acher, and B. Baudry, **"From comparison matrix to Variability Model: The Wikipedia case study,"** in 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings, 2013, pp. 580–585.

[33] J. Rubin and M. Chechik, **"Locating distinguishing features using diff sets,"** Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on, 2012. [Online]. Available: http://www.cs.toronto.edu/~chechik/pubs/ase12.pdf. [Accessed: 04-Apr-2015].

[34] S. Deelstra and M. Sinnema, **"Managing the complexity of variability in software product families,"** Doctoral Thesis , 2008. [Online]. Available: http://dissertations.ub.rug.nl/faculties/science/2008/m.sinnema/?pLanguage=en&pFullItemRecord=ON. [Accessed: 27-Apr-2015].

[35] K. Czarnecki and M. Antkiewicz, **"Mapping features to models: A template approach based on superimposed variants,"** in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2005, vol. 3676 LNCS, pp. 422–437.

[36] B. Zhang and M. Becker, **"Mining complex feature correlations from software product line configurations,"** in Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, 2013, p. 19.

[37] R. Al-Msie'Deen, a. D. Seriai, M. Huchard, C. Urtado, and S. Vauttier, **"Mining features from the object-oriented source code of software variants by combining lexical and structural similarity,"** in Proceedings of the 2013 IEEE 14th International Conference on Information Reuse and Integration, IEEE IRI 2013, 2013, pp. 586–593.

[38] J. Kofroň, F. Plášil, and O. Šerý, **"Modes in component behavior specification via EBP and their application in product lines,"** Inf. Softw. Technol., vol. 51, no. 1, pp. 31–41, Jan. 2009.

[39] R. Koschke and J. Quante, **"On dynamic feature location,"** in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering SE - ASE '05, 2005, pp. 86–95.

[40] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, **"On extracting feature models from product descriptions,"** in Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '12, 2012, pp. 45–54.

[41] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, **"On extracting feature models from sets of valid feature combinations"**, vol. 7793. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[42] J. Feigenspan, **"Program Comprehension of Feature-Oriented Software Development."** , University of Magdeburg, Germany, [Online]. Available: http://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/auto/Feigenspan11.pdf. [Accessed: 20-Mar-2015].

[43] L. P. Tizzei, C. M. F. Rubira, **"Public Health Complaint Software Product Line,"** 2011.

[44] B. Zhang and M. Becker, **"RECoVar: A solution framework towards reverse engineering variability,"** in 2013 4th International Workshop on Product LinE Approaches in Software Engineering, PLEASE 2013 - Proceedings, 2013, pp. 45–48.

[45] A. Hamou-Lhadj, E. Braun, D. Amyot, and T. Lethbridge, **"Recovering behavioral design models from execution traces,"** in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2005, pp. 112–121.

[46] H. Eyal-Salman, A.-D. Seriai, C. Dony, and R. Al-msie'deen, **"Recovering traceability links between feature models and source code of product variants,"** in Proceedings of the VARiability for You Workshop on Variability Modeling Made Useful for Everyone - VARY '12, 2012, pp. 21–25.

[47] S. Duszynski and M. Becker, **"Recovering variability information from the source code of similar software products,"** 2012 3rd Int. Work. Prod. LinE Approaches Softw. Eng. PLEASE 2012 - Proc., pp. 37–40, Jun. 2012.

[48] A. Olszak and B. Nørregaard Jørgensen, **"Remodularizing Java programs for improved locality of feature implementations in source code,"** Sci. Comput. Program., vol. 77, no. 3, pp. 131–151, Mar. 2012.

[49] F. Loesch and E. Ploedereder, **"Restructuring variability in software product lines using concept analysis of product configurations,"** Proc. Eur. Conf. Softw. Maint. Reengineering, CSMR, pp. 159–168, 2007.

[50] D. Kim, W. N. Sumner, X. Zhang, D. Xu, and H. Agrawal, **"Reuse-oriented reverse engineering of functional components from x86 binaries,"** in Proceedings of the 36th International Conference on Software Engineering - ICSE 2014, 2014, pp. 1128–1139.

[51] L. D. Mathieu Acher, Anthony Cleve, **"Reverse Engineering Architectural Feature Models,"** Proc. 5th Eur. Conf. Softw. Archit. (ECSA'11}, 2011.

[52] B. Zhang and M. Becker, **"Reverse Engineering Complex Feature Correlations for Product Line Configuration Improvement,"** in 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, 2014, pp. 320–327.

[53] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, **"Reverse engineering feature models,"** in 2011 33rd International Conference on Software Engineering (ICSE), 2011, pp. 461–470.

[54] E. N. Haslinger, **"Reverse Engineering Feature Models from Program Configurations,"** Master's thesis, 2012. [Online]. Available: http://www.jku.at/JKU_Site/JKU/isse/content/e104861/e105613/e179647/e179648/DA_Haslinger.pdf. [Accessed: 04-Apr-2015].

[55] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, **"Reverse engineering feature models from programs' feature sets,"** in Proceedings - Working Conference on Reverse Engineering, WCRE, 2011, pp. 308–312.

[56] R. A.-M. 'Deen, M. Huchard, A.-D. Seriai, C. Urtado, and S. Vauttier, **"Reverse Engineering Feature Models from Software Configurations using Formal Concept Analysis,"** CLA 2014: Eleventh International Conference on Concept Lattices and Their Applications, vol. 1252. pp. 95–106, 01-Oct-2014.

[57] G. Bécan, **"Reverse Engineering Feature Models in the Real."** Master's thesis, INRIA-IRISA Rennes Bretagne Atlantique, équipe TRISKELL, p. 40, 25-Jun-2013.

[58] A. E. Roberto E. Lopez-Herrejon1, José A. Galindo2, David Benavides2, Sergio Segura2, **"Reverse engineering feature models with evolutionary algorithms: an exploratory study"**, vol. 7515. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[59] J. Knodel, R. Koschke, and T. Mende, **"Reverse engineering in a reuse context."** Kaiserslautern, 2006.

[60] E. K. Abbasi and P. Heymans, **"Reverse engineering web sales configurators,"** in IEEE International Conference on Software Maintenance, ICSM, 2014, pp. 586–589.

[61] K. Czarnecki, S. She, and A. Wasowski, **"Sample spaces and feature models: There and back again,"** in Proceedings - 12th International Software Product Line Conference, SPLC 2008, 2008, pp. 22–31.

[62] M. Acher, B. Baudry, P. Heymans, A. Cleve, and J.-L. Hainaut, **"Support for reverse engineering and maintaining feature models,"** in Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, 2013, p. 20.

[63] J. Feigenspan, M. Schulze, M. Papendieck, C. Kastner, R. Dachselt, V. Koppen, M. Frisch, and G. Saake, **"Supporting program comprehension in large preprocessor-based software product lines,"** IET Softw., vol. 6, no. 6, p. 488, 2012.

[64] G. Bécan, R. Behjati, A. Gotlieb, and M. Acher, **"Synthesis of Attributed Feature Models From Product Descriptions: Foundations,"** Inria Rennes, Feb. 2015.

[65] T. Ziadi, C. Henard, M. Papadakis, M. Ziane, and Y. Le Traon, **"Towards a Language-Independent Approach for Reverse-Engineering of Software Product Lines,"** in Sac '14, 2014, pp. 1064–1071.

[66] L. Moonen, **"Towards evidence-based recommendations to guide the evolution of component-based product families,"** Sci. Comput. Program., vol. 97, pp. 105–112, Jan. 2013.

[67] J. Feigenspan and M. Schulze, **"Using background colors to support program comprehension in software product lines,"** in International Conference on Evaluation and Assessment in Software Engineering, 2011, pp. 66–75.

[68] I. Pashov and M. Riebisch, **"Using feature modeling for program comprehension and software architecture recovery,"** Proc. - 11th IEEE Int. Conf. Work. Eng. Comput. Syst. ECBS 2004, pp. 406–417, May 2004.

[69] C. Kastner, A. Dreiling, and K. Ostermann, **"Variability Mining: Consistent Semi-automatic Detection of Product-Line Features,"** IEEE Trans. Softw. Eng., vol. 40, no. 1, pp. 67–82, Jan. 2014.

[70] S. Duszynski, **"Visualizing and analyzing software variability with bar diagrams and occurrence matrices,"** Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. , vol. 6287 LNCS, pp. 481–485, Sep. 2010.

[71] O. Greevy, M. Lanza, and C. Wysseier, **"Visualizing feature interaction in 3-D,"** in Proceedings - VISSOFT 2005: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2005, pp. 114–119.

[72] G. Bécan, S. Nasr, M. Acher, and B. Baudry, **"WebFML: Synthesizing Feature Models Everywhere,"** SPLC-18th International Software Product Line Conference pp. 1–5, 15-Sep-2014.

3 Phase 3 – schémas de classification et évaluation de la qualité

La phase 3 est séparée en 2 sous-phases. Une première sous phase permet la création d'un schéma de classification. La seconde sous phase établit une mesure de qualité de la littérature retenue.

3.1 Phase 3 - Extraction de mots-clés et schéma de classification

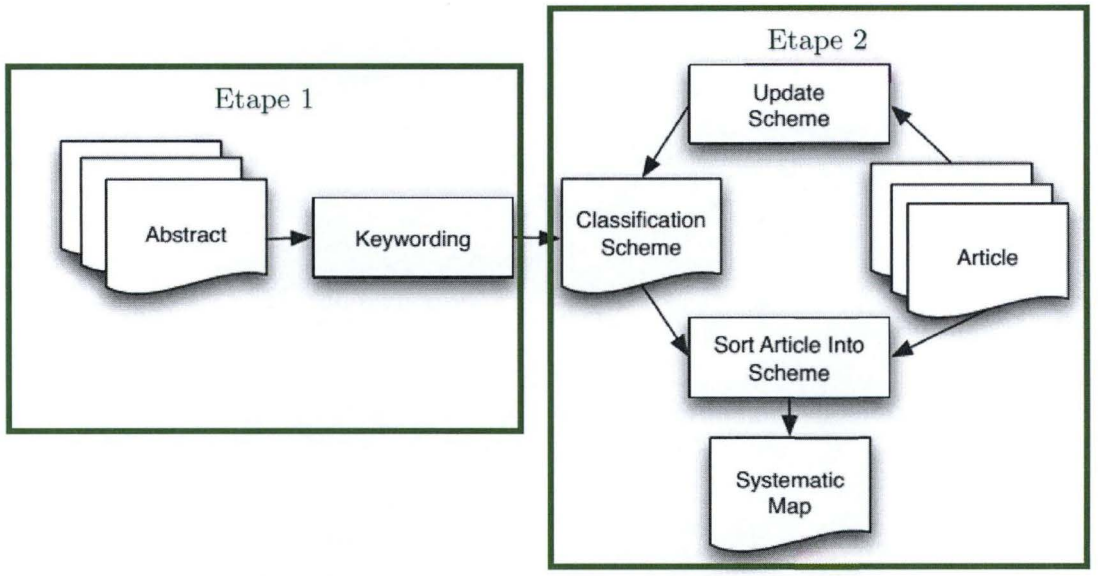


Figure 27 - Building the Classification Schema (Petersen, et al. s.d.)

Une fois les différentes publications récupérées et triées, un schéma de classification est créé. Pour ce, un processus en 2 étapes est appliqué (Devroey, Perrouin, et al., 2014) et basé sur la Figure 27 ci-dessus.

Contrairement au protocole fixé dans la Figure 18,

I. chaque publication est représentée par des mots-clés représentant le contenu du document (extraits de l'abstract ou de l'introduction / conclusion ou encore d'une lecture sommaire des passages concernant le sujet au besoin tel que décrit dans « Adaptive Reading Depth For Classification » (Petersen, et al. s.d.)). Afin d'être au maximum en adéquation avec le sujet de la mapping study, les mots-clés seront orientés suivant les questions de recherche (ex. : dans le cas du REV, des mots-clés orientés « type de modèle d'entrée » seraient orientés recherche de réponse à la question QR2)[24]. Il va aussi de soi que les potentielles parties non pertinentes présentes dans les publications ne seront pas reprises (ex. : une technique (QR1) présentée dans une thèse (QR6) ayant une source d'entrée (QR2QR1) et source de sortie (QR3) concernant une rétro-ingénierie n'ayant aucun rapport ni de près ni de loin avec la RIV ou d'interprétation du comportement permettant l'interprétation de la variabilité, les mots-clés correspondants aux questions de recherche ne seront pas pris en compte).

[24] Cette génération de mots-clés n'est donc pas automatisable via des outils tels que Terminology Extraction (Translated Labs s.d.), Extractor (Technologies, DBI s.d.), jatetoolkit (ziqizhan...@googlemail.com s.d.) ... le niveau d'abstraction étant trop important et le but n'étant pas de chercher des mots-clés techniques d'un document.

La plupart des publications scientifiques contiennent déjà une série de mots-clés caractérisant la publication. Ces mots-clés peuvent êtres (en partie/totalement) repris dans cette classification. Cependant, ils ne sont pas suffisants sachant qu'il s'agit aussi de classifier les documents par rapport à leurs contenu ET aux QRs (ce dont les mots-clés d'auteurs ne sont pas nécessairement orienté).

II. en regroupant et combinant les différents mots-clés dans des catégories de haut niveau (représentable sous la forme d'arbres). Les catégories analogues sont regroupées et appelées facettes.

> Une **Facette** est un aspect défini permettant de regrouper un ensemble de publications au sein d'une « systematic mapping study ».

3.2 Résultats de la phase 3

Afin de résumer l'étape 1 de manière la plus lisible possible, les mots-clés ont été triés puis fusionnés en mots-clés très légèrement plus génériques. Ces mots-clés sont ensuite synthétisés via la seconde 2 du processus en facettes.

Ci-après, se trouvent les résultats des 2 étapes sous différentes formes dont beaucoup sous la forme d'arbres dont les nœuds rouges représentent les facettes et les nœuds bleus représentent les mots-clés fusionnés. Ce type de représentation permet, d'une part de faire ressortir les facettes mais en plus, de donner un maximum de détails et d'informations sur les mots-clés synthétisés derrière ces facettes. Afin d'éviter toute redondance non pertinente, chaque mot-clé renseigne la ou les publications qu'il représente. Ainsi, par exemple, l'article [4] « A Semi-Automatic Approach for Extracting Software Product Lines » est représenté par les mots-clés « approach » et « case study » étant représentés par deux facettes différentes.

Après avoir effectué cette 3ème phase, les facettes suivantes ont étés construites sous forme d'arbres détaillés ci-après :

Entry source (QR2)

Facettes représentant les sources d'entrée des techniques de RIV.

- « Code source »
- « Diagram / model »
- « Informal »
- « Feature sets »
- « Propositional logic »
- « Ontological semantics »
- « Execution Traces »
- « Matrix »

Output artifacts (QR3)

Facettes représentant les artéfacts de sortie des techniques de RIV.

- « Models »
- « Logical constraints »
- « Code region has potentially feature »
- « Code source annotation »
- « Background color »
- « Features cohesion »
- « Product variant »
- « Feature correlations »
- « Software Product Line (SPL) »
- « Variability-related facts »
- « Feature traces »

Variability manager mechanism (QR4)

Facettes représentant les différents mécanismes permettant la gestion (expression et manipulation) de la variabilité au sein des techniques de RIV.

- « Annotations and informations from source code »
- « Merging set of product description »
- « Behavior specification »
- « Models »
- « Language »
- « Latent semantic indexing (LSI) »
- « Cross-tree constraints (CTCs) »
- « Protocol »
- « Formal/Relational Concept Analysis »
- « Conditional Compilation (CC) mechanism »

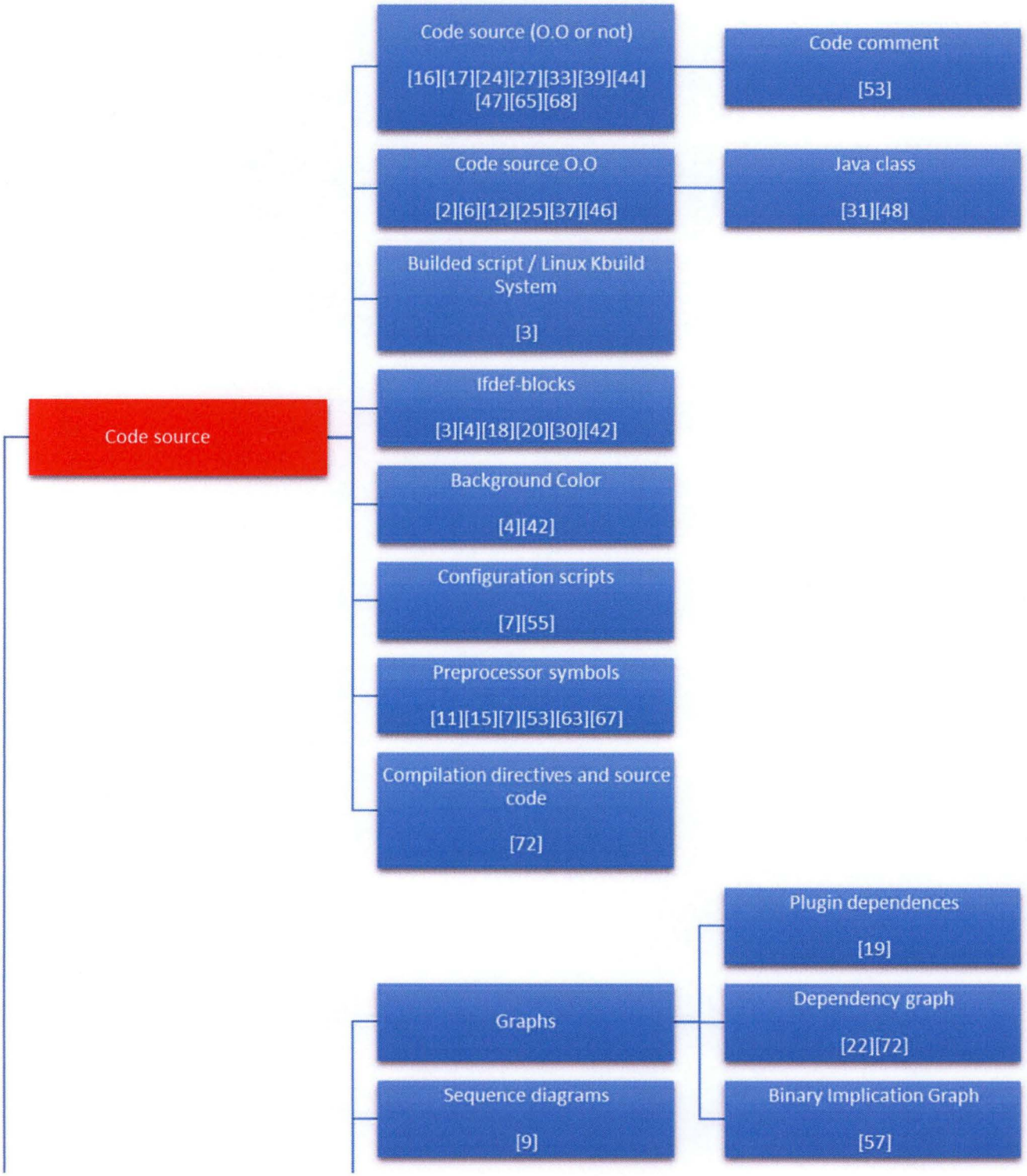
Intermediary artifact (QR4 – sous question)

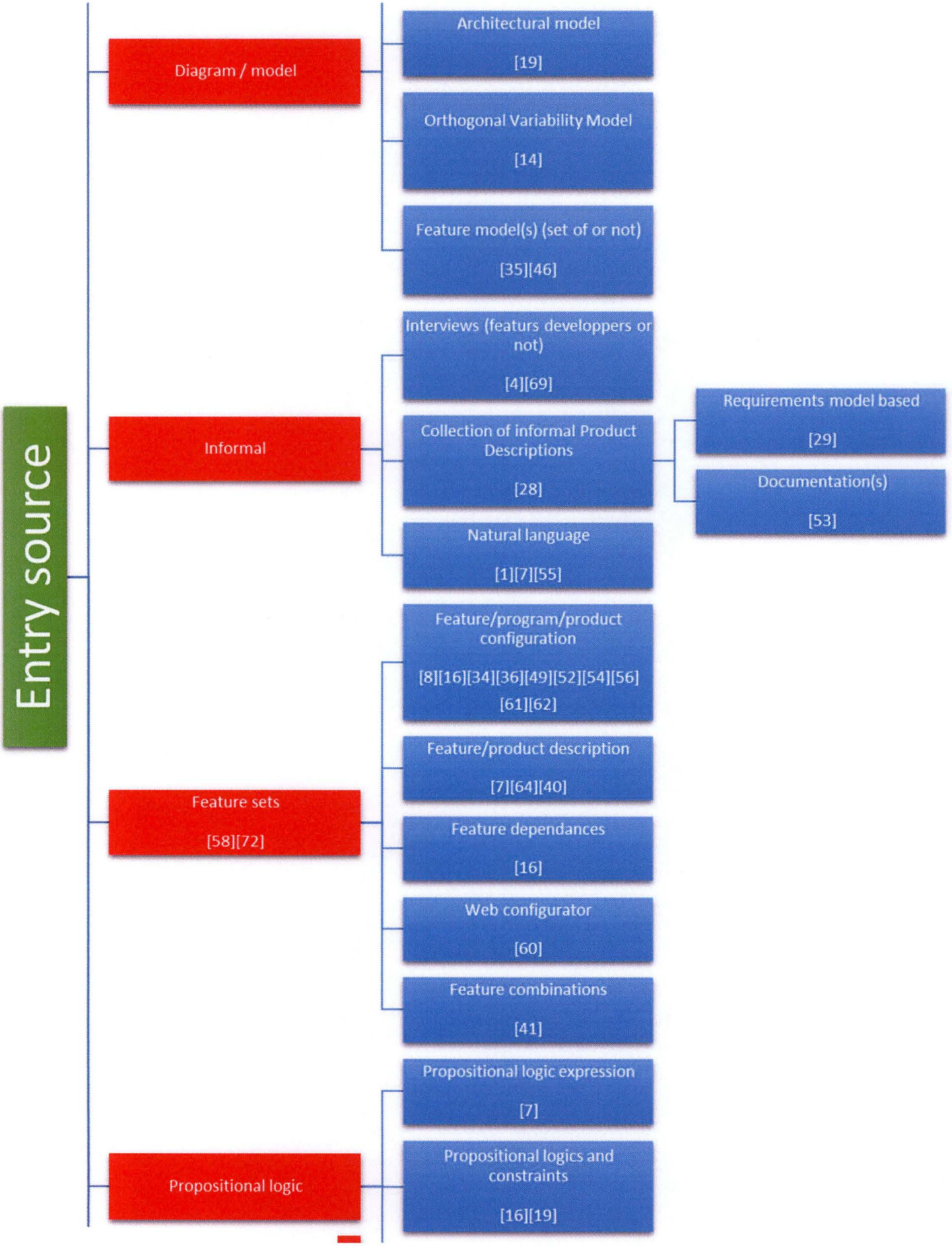
Facettes représentant les artéfacts intermédiaires utilisés par les techniques de RIV

- « Concept analysis »
- « Latent semantic indexing (LSI) »
- « Common object-oriented building elements (COBE) »
- « Algorithm »
- « Formulas »
- « Models »
- « Graph »
- « Tree »
- « Legal joint probability distributions (JPDs)"
- « Language »
- « 150% architecture »

I. Facettes « Entry source » (QR2)

Les différentes techniques, approches et algorithmes recensés requièrent tous une ou des sources d'entrée sur lesquelles ils se basent. Ces sources sont reprises et regroupées dans la Figure 28 ci-après.




Entry source
Diagram / model
Architectural model
[19]
Orthogonal Variability Model
[14]
Feature model(s) (set of or not)
[35][46]
Informal
Interviews (featurs developpers or not)
[4][69]
Collection of informal Product Descriptions
[28]
Requirements model based
[29]
Documentation(s)
[53]
Natural language
[1][7][55]
Feature sets
[58][72]
Feature/program/product configuration
[8][16][34][36][49][52][54][56]
[61][62]
Feature/product description
[7][64][40]
Feature dependances
[16]
Web configurator
[60]
Feature combinations
[41]
Propositional logic
Propositional logic expression
[7]
Propositional logics and constraints
[16][19]

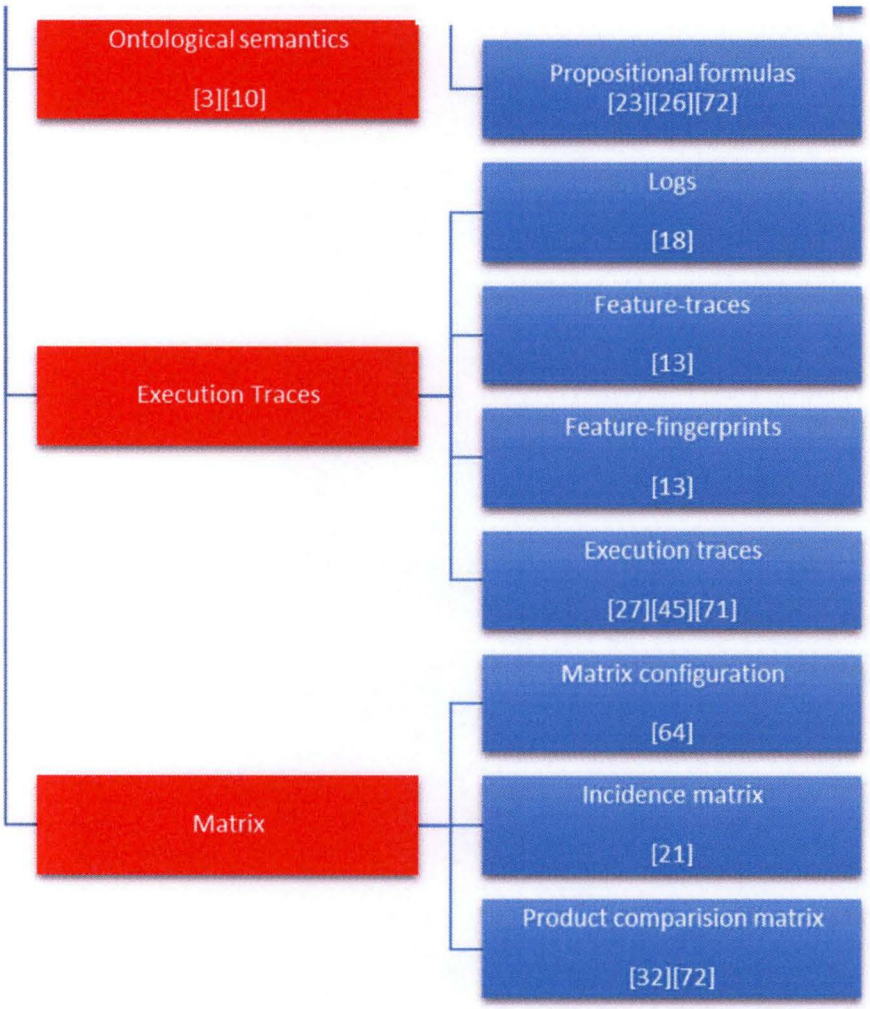


Figure 28 – Arbre des facettes "Entry source"

II. Facettes « Output artefact » (QR3)

Les approches et techniques reprises dans les publications produisent différents artéfacts de sortie. Ces artéfacts sont exprimés et regroupés dans la Figure 29 ci-après.

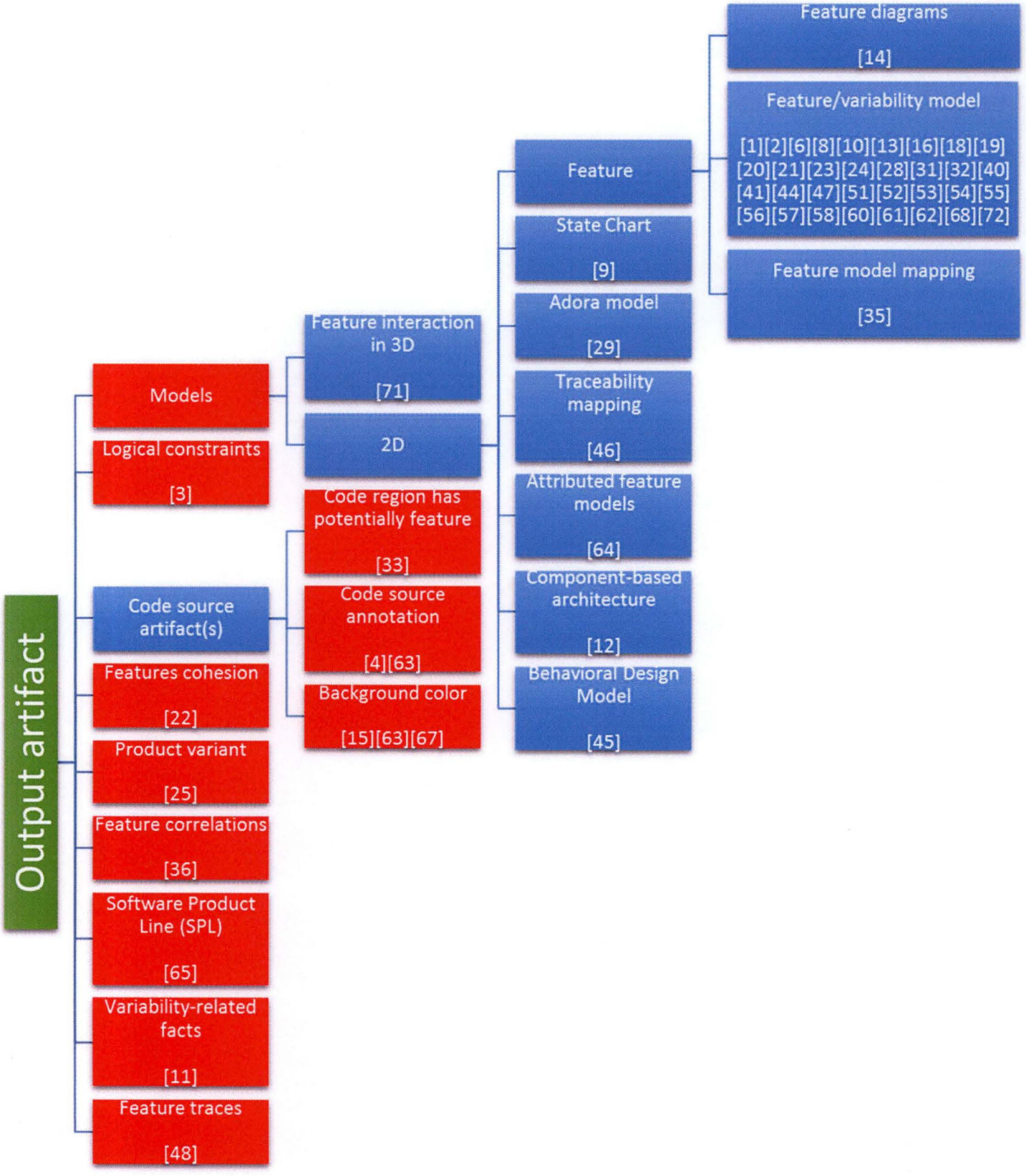
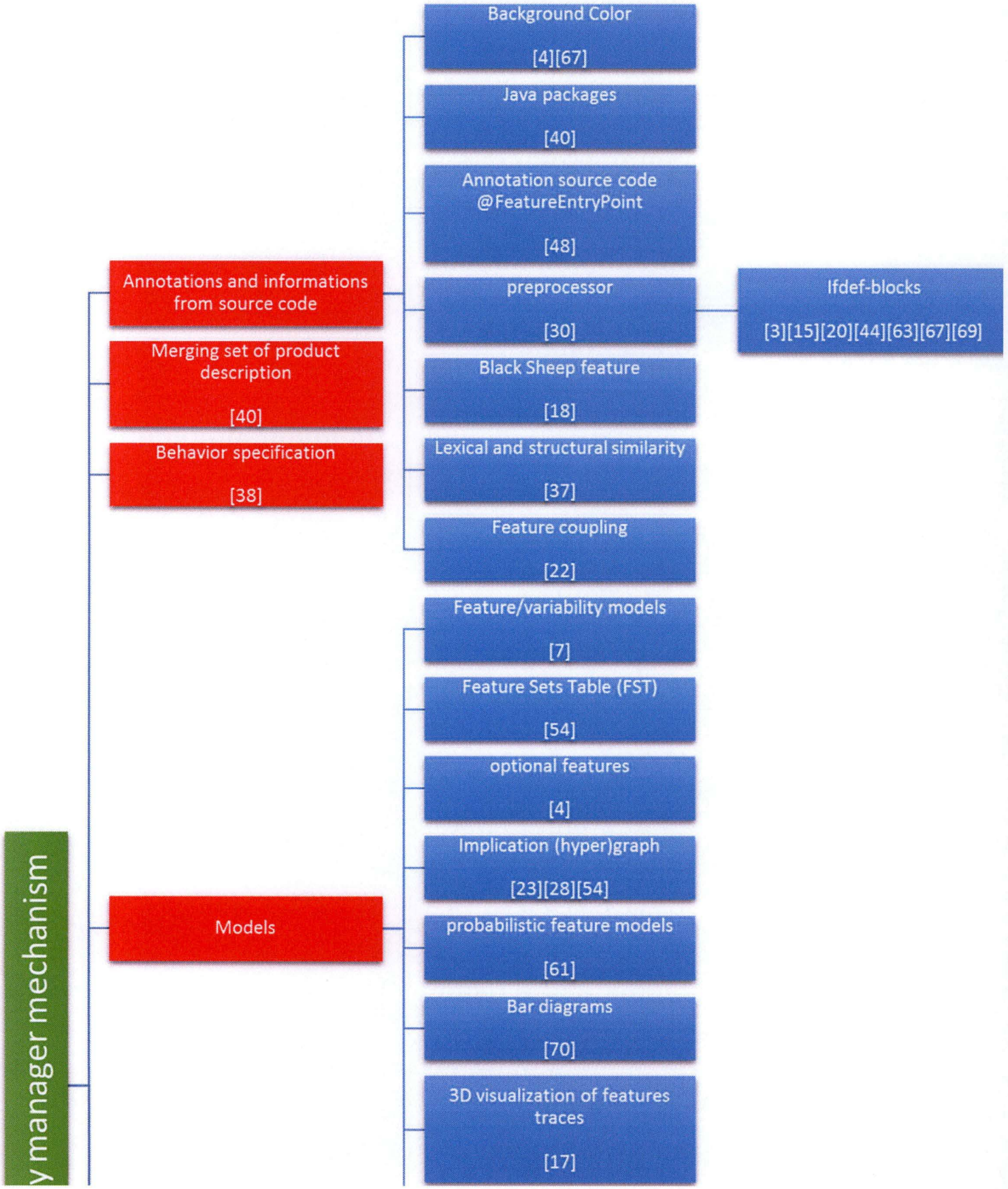


Figure 29 - Arbre des facettes "Output artifact"

III. Facettes « Variability manager mechanism » (QR4)

Les facettes de la Figure 30 ont pour but d'exprimer et de regrouper les différents mécanismes cités permettant la gestion de la variabilité dans ces techniques.



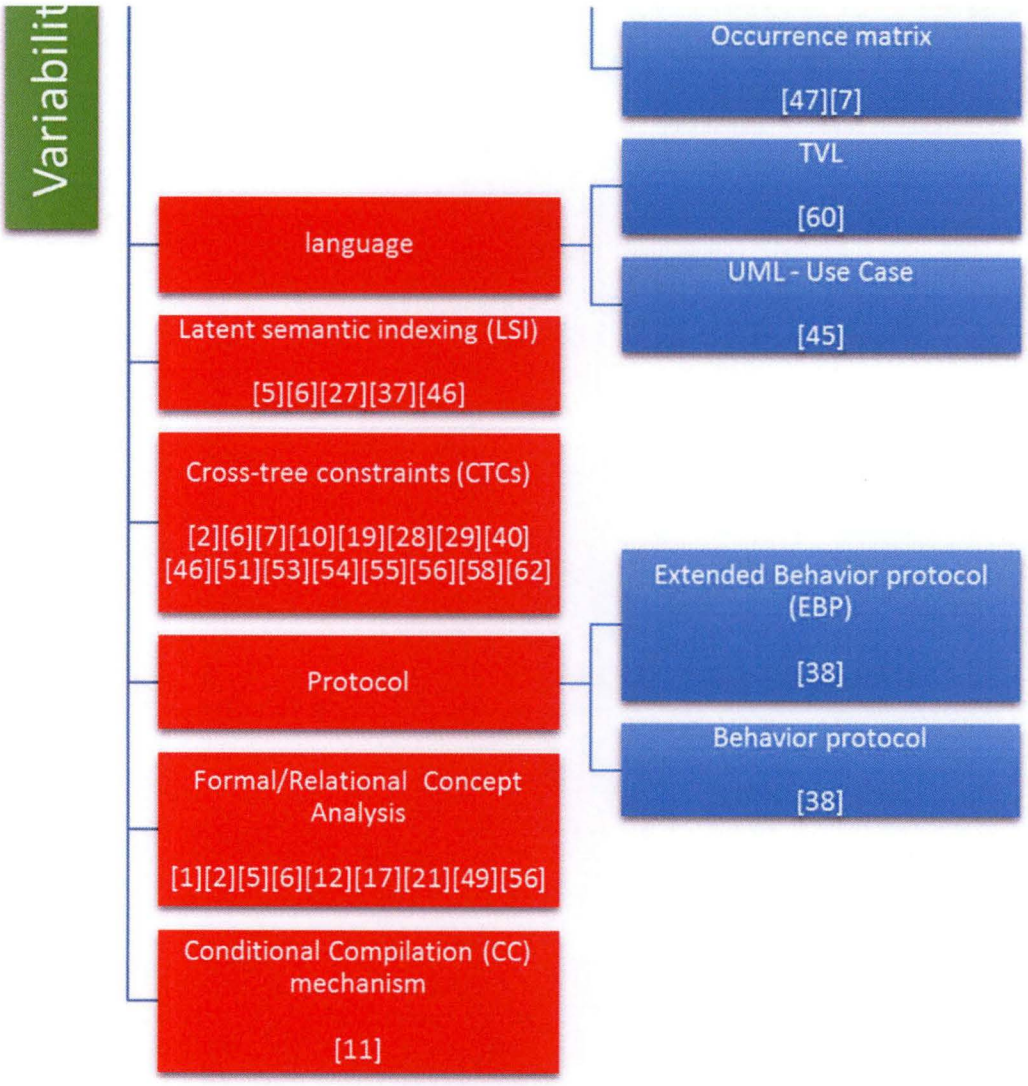
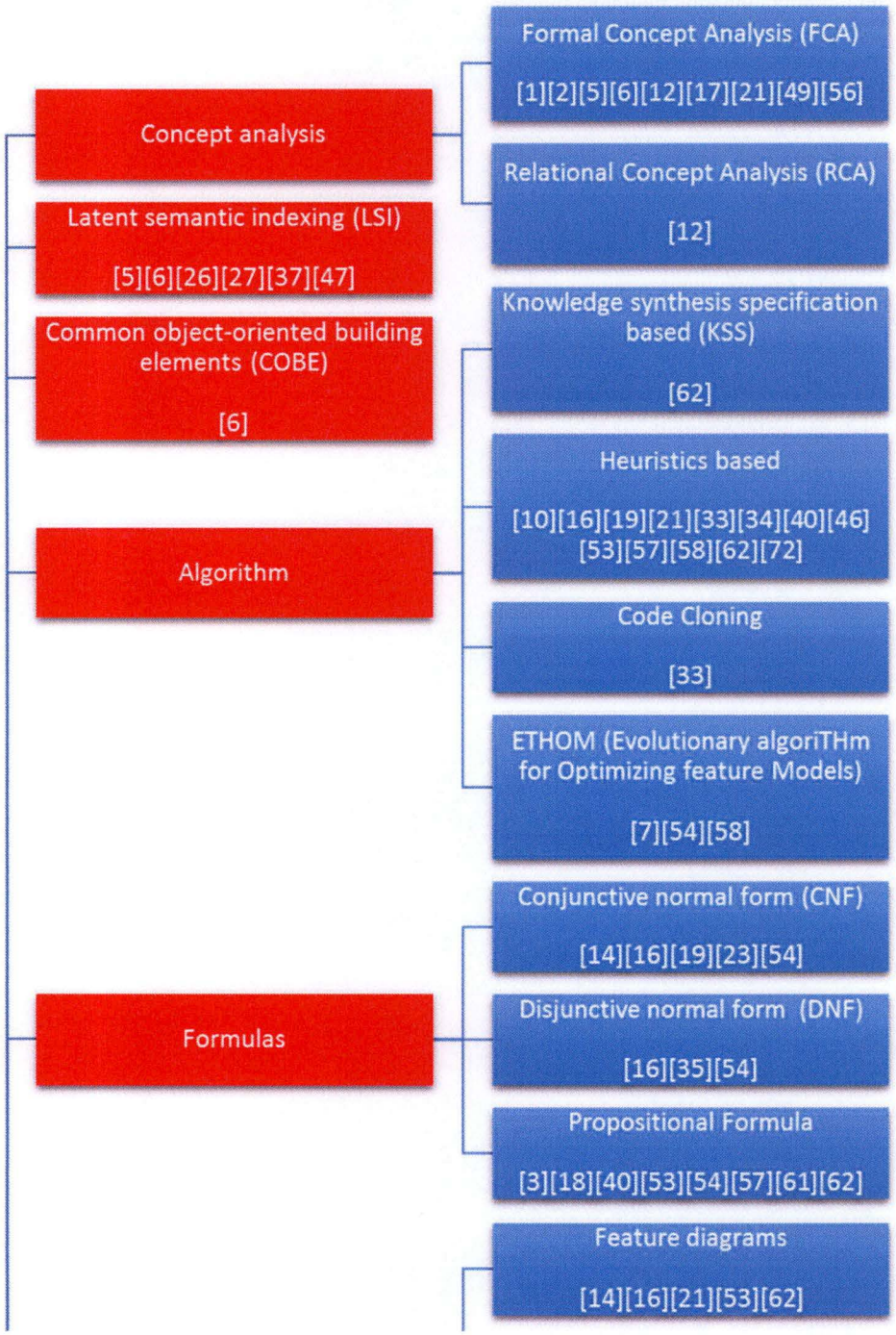
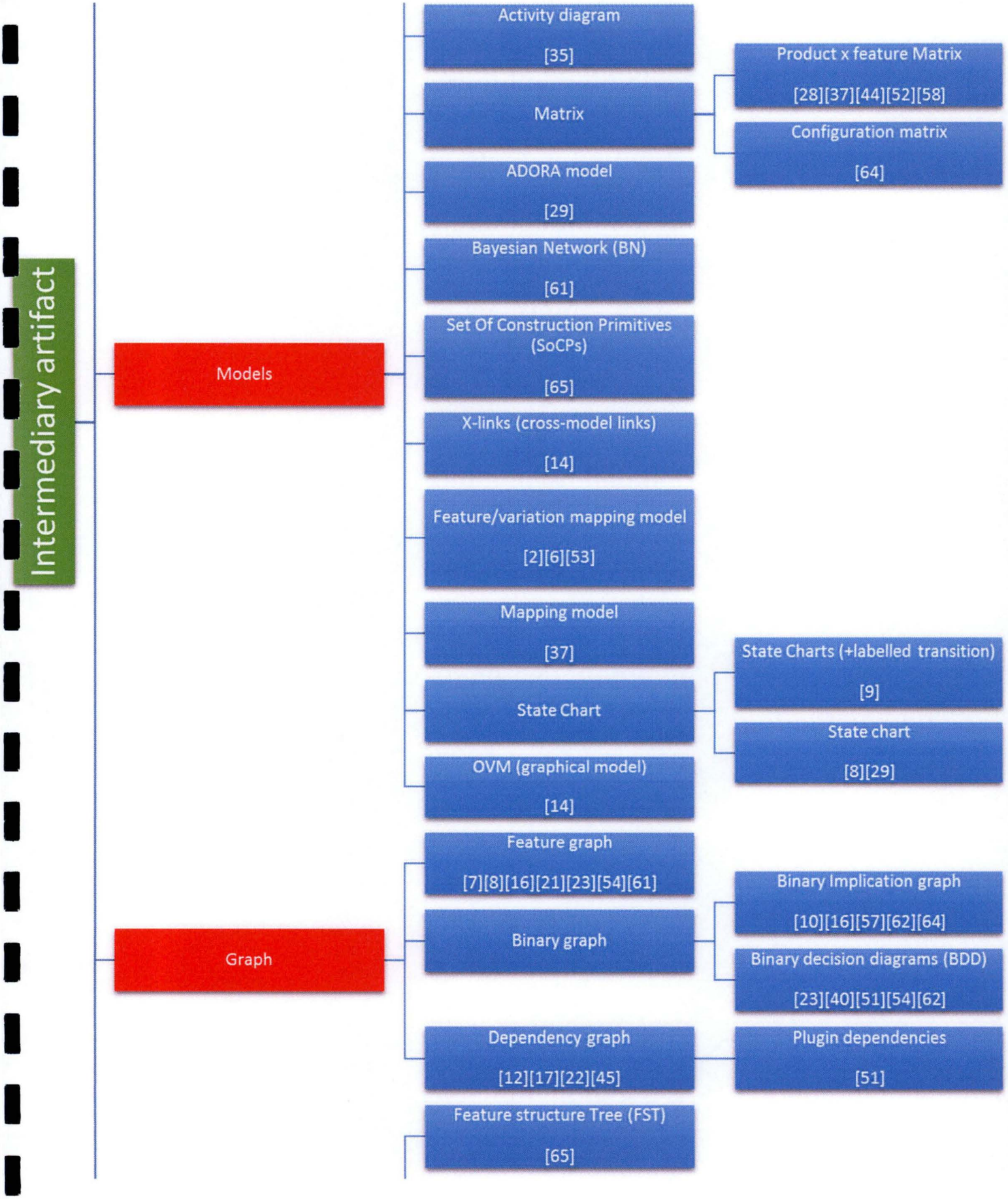


Figure 30 - Arbre des facettes "Variability manager mecanism"

IV. Facettes « Artifacts intermediaries » (QR4 – sous question)

Les facettes de la Figure 31 ont pour but d'exprimer les différents artéfacts intermédiaires (modèles, langages, algorithmes, formules) utilisés dans les techniques récupérées. Ce sont des artéfacts utilisés dans le cadre et d'une et des parties précises des techniques, ce ne sont ni les entrées ni les sorties de techniques. Ils servent juste de « tremplin » permettant de mener à terme la technique.




Intermediary artifact
Models
Activity diagram
[35]
Matrix
Product x feature Matrix
[28][37][44][52][58]
Configuration matrix
[64]
ADORA model
[29]
Bayesian Network (BN)
[61]
Set Of Construction Primitives (SoCPs)
[65]
X-links (cross-model links)
[14]
Feature/variation mapping model
[2][6][53]
Mapping model
[37]
State Chart
State Charts (+labelled transition)
[9]
State chart
[8][29]
OVM (graphical model)
[14]
Graph
Feature graph
[7][8][16][21][23][54][61]
Binary graph
Binary Implication graph
[10][16][57][62][64]
Binary decision diagrams (BDD)
[23][40][51][54][62]
Dependency graph
[12][17][22][45]
Plugin dependencies
[51]
Feature structure Tree (FST)
[65]

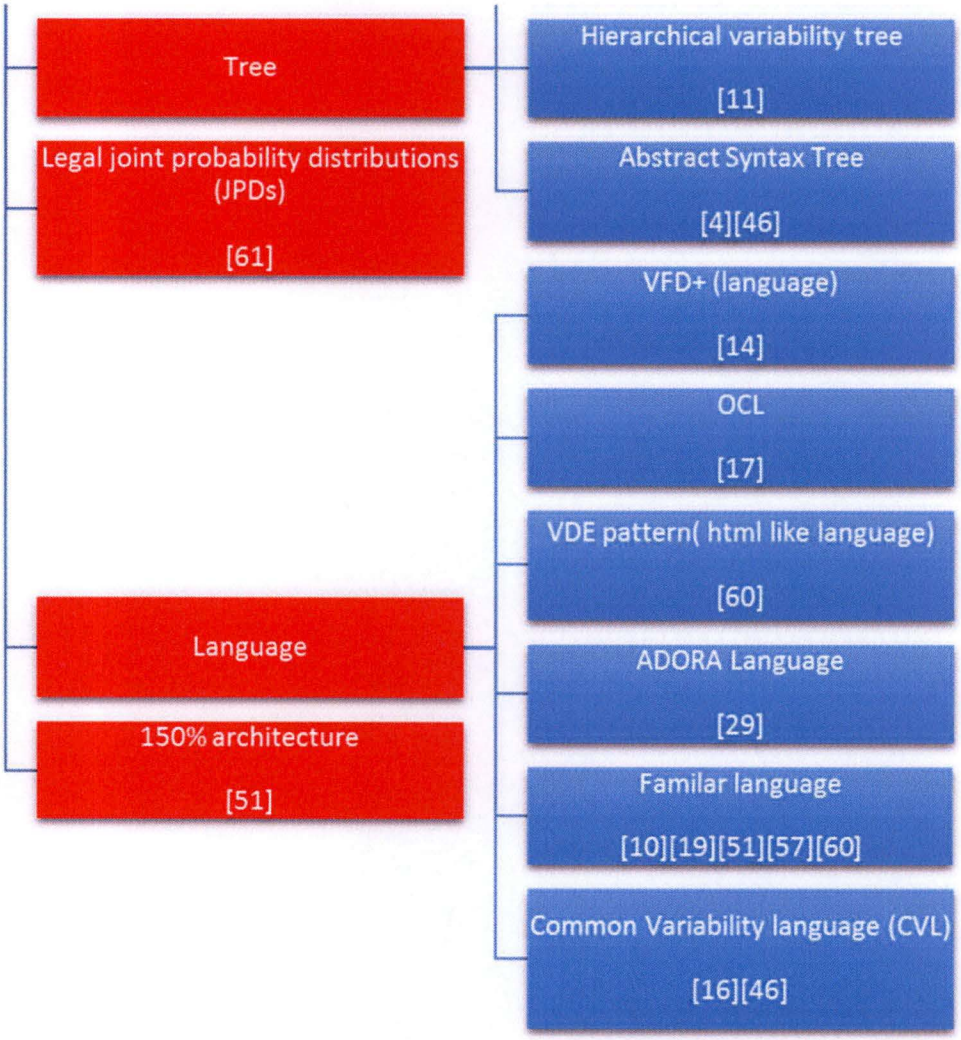


Figure 31 - Arbre des facettes "Intermediary artifact"

V. Facettes « Technique d'évaluation de la couverture » (QR5)

Lorsqu'une technique est appliquée, il est évidement intéressant de déterminer si oui ou non elle répond totalement à la demande. Pour ce, 4 facettes sont explicitement exprimées. La Figure 32 ci-après reprend ces 4 « techniques ».

La forme qu'a cette figure a une signification particulière et doit être interprétée comme suit : « Chaque facette peut être utilisée séparément ou couplée avec une ou plusieurs autres ».

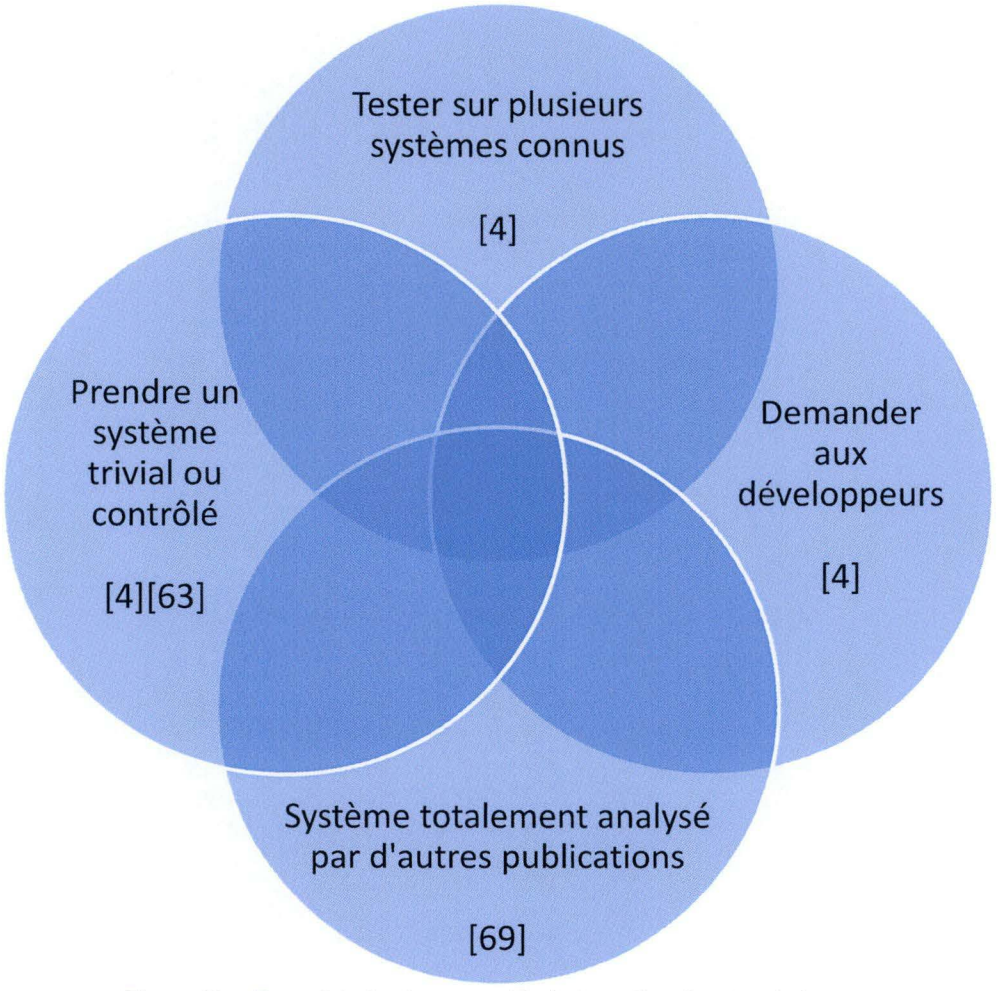


Figure 32 – Ensemble des facettes « Technique d'évaluation de la couverture »

VI. Facettes « Publication type » (QR6)

Ces facettes ont pour but d'exprimer quels sont les différents types de littératures et de les regrouper en facette prédéfinies.

Mis à part la facette « Thesis », les autres facettes proposées proviennent de la ligne de conduite proposée dans Kai Petersen et al (Petersen, et al. s.d.). Ci-après se trouve le tableau traduit proposé dans cette susmentionnée ligne de conduite. Suivre cette ligne de conduite au niveau type de publications n'est pas anodin, cela donne un plus dans le cas de comparaisons de mapping study. En effet, si quelqu'un voulait comparer cette mapping study avec une autre suivant aussi cette ligne de conduite (ex. : dans le but de comparer le type de littératures existant dans 2 domaines (proches ou éloignés)), il pourrait très facilement les comparer sachant que les facettes seront identiques. Cela revient quelque peu à appliquer un standard réutilisable par tous. Ce principe n'est hélas pas applicable pour les autres facettes car elles sont trop orientées QRs.

Tableau 5 - Facettes « type de document » (Petersen, et al. s.d.) (Roel Wieringa 2006)

Catégorie	Description
Recherche de validation	Les techniques étudiées sont nouvelles et ne sont pas encore mises en œuvre dans la pratique. Les techniques utilisées sont, par exemple, expérimentales, sortant de laboratoire.
Recherche d'évaluation	les techniques sont mises en œuvre dans la pratique et une évaluation de la technique est utilisée. Cela signifie qu'il est montré comment la technique et mise en œuvre dans la pratique (solution d'implémentation) et quelles sont les conséquences de la mise en œuvre en termes d'avantages et inconvénients (évaluation de la mise en œuvre). Cela inclut également l'identification de problèmes dans l'industrie.
Proposition de solution	Une solution à un problème est proposée, la solution peut être soit nouvelle ou être une extension significative d'une technique existante. Les potentiels avantages et l'applicabilité de la solution sont présentés par un exemple ou un ensemble d'argumentations.
Documents philosophiques	Ces documents esquissent une nouvelle façon de voir les choses existantes en structurant la forme de la taxonomie ou du cadre conceptuel.
Documents d'opinions	Ces documents expriment l'opinion personnelle de quelqu'un mentionnant si une certaine technique est bonne ou mauvaise, ou comment les choses devraient être faites. Ils ne se reposent pas sur des travaux connexes et méthodologies de recherche.
Document d'expérience	Un document d'expérience expliquant sur quoi et comment quelque chose a été fait dans la pratique. Ça doit être l'expérience personnelle de l'auteur.

VII. La facette « Outils » (QR7)

Cette facette reprend, sous forme de liste, les différents outils mentionnés/expliqués dans les publications. Ceux-ci sont décrits lors de l'analyse des outils dans la Phase 4.

Tableau 6 – Tableau de la facette "Outils"

Outil	Publications	Outil	Publications	Outil	Publications
Adora tool :	[29]	AHEAD tool suite	[24]	BeTTy Framework	[5][7][58]
CCVisu	[22]	CIDE	[4]	ETHOM	[7]
FAMA tool suite	[7][54][55][56][62]	Familiar	[10][19][51][57][60]	Feature Commander	[20][30][63]
FeatureVisu	[22]	Fmp2rsm plugin	[35]	Framework RECoVar	[44]
Golem tool	[3]	Javapp	[18]	KBuildMiner	[3]
LEADT tool (java code)	[69]	Orange tool	[44]	REVPLINE	[56]
WebFML	[10]				

3.3 Phase 3 bis - Evaluation de la qualité

Comme suggéré par da Mota Silveira Neto et al. (Paulo Anselmo da Mota Silveira Neto, et al. 2010), il convient de mesurer la qualité de chaque publication récupérée. Bien que ce mécanisme d'évaluation de la qualité soit souvent utilisé dans le cadre de « literature reviews », il est aussi appliqué dans cette mapping study afin de mesurer la fiabilité des études menées dans les différents documents récupérés. Cette étape est nécessaire afin de limiter les différents préjugés du meneur de cette étude ainsi que de comprendre les comparaisons faites (ex. : préjugé sur les auteurs, les revues scientifiques comme ACM/IEEE/SIAM, provenance d'une université ou centre de recherche ...).

Les différents critères de qualité repris ci-après sont donc des moyens objectifs de déterminer une pondération sur l'importance de chacun des documents.

Dans cette mapping study, 4 groupes de critères de qualité sont retenus :

- Le 1er groupe reprend des critères concernant la qualité de la forme de la publication.
- Le 2e groupe permet les critères de qualité d'une publication sur base de la profondeur de l'étude menée dans cette dernière.
- Le 3e groupe donne une idée de la qualité de la ou des techniques proposées et sur leurs critiques.
- Le 4e groupe reprend un critère de qualité permettant d'avoir une idée sur la technique en elle-même et plus précisément sur son efficacité supposée.

Tableau 7 - Critères de qualité de publications

Groupe	Identifiant	Critère de qualité
1	G 1.1	Contient-il une motivation ? Explique-t-il l'utilité d'une telle technique ?
	G 1.2	Contient-il une ligne de conduite sur comment le RIV peut-il être appliqué ?
	G 1.3	La publication contient-elle un paragraphe reprenant les « threats to validity » ?
	G 1.4	Contient-il un référentiel (explicite ou implicite) des techniques existantes ?
2	G 2.1	Fait-il une comparaison entre différentes techniques (détaillée ou non) ?
	G 2.2	Reprend-il les points forts, faibles et/ou limites des/de la technique(s) étudiée(s) ?
	G 2.3	Explique-t-il en détail une technique de RIV précise ?
	G 2.4	Est-il comparatif dans les techniques de RIV détaillées ?
	G 2.5	Contient-il au moins une étude de cas [25]?
3	G 3.1	Explique-t-il précisément les entrées et sorties des différentes techniques[26] ?
	G 3.2	Propose-t-il un(e) outil/technique applicable hors du milieu de la recherche ?
	G 3.3	La technique de RIV est-elle orientée web ?

[25] Il s'agit bien d'une étude de cas (considéré comme telle) sur un sujet existant. Il ne s'agit nullement de la présence d'un exemple créé exclusivement pour l'explication d'une méthode.
[26] Il est sous-entendu : un « artéfact de documentation » en entrée n'est pas une source d'entrée précise. Le « code source O.O » d'une application est une source d'entrée précise.

	G 3.4	Propose-t-il un(e) outil/méthode fonctionnelle[27] ?
	G 3.5	Propose-t-il un(e) outil/méthode partiellement fonctionnelle [28] ?
4	G 4.1	La technique permet-elle d'exprimer la totalité de la variabilité d'un système ?

L'interprétation des résultats du Tableau 8 doit se faire en comparant les qualités présentes dans les publications et non celles non présentes. En effet, certaines qualités n'ont pas d'application sur certaines publications (ex. : une litterature reprenant une liste de techniques détaillées n'a pas lieu d'être classée en fonction de la qualité 4.1).

(V = présent, ~ = présent mais par explicite)

Tableau 8 - Tableau de classification de publications suivant les critères de qualité

Critère publication	G1.1	G1.2	G1.3	G1.4	G2.1	G2.2	G2.3	G2.4	G2.5	G3.1	G3.2	G3.3	G3.4	G3.5	G4.1
1*		v			v						v		v		
2	v	v		v			v			v				v	v
3	v	v	v	~	v	v	v		v	v	~			v	v
4		v	v	v	v	v	v		v	v	v		v		~
5				v	v			v		v					
6	v	v		~	v					v				v	
7			v	v	v			v		v					
8	v								v						v
9		v					v			v	v			v	
10	v		v	v	v	v	v			v				v	
11*									v	v					
12	v	v		v			v			v				v	
13	v	v		v		v	v		v	v				v	
14	v	v					v			v				v	
15	v	v	v	v		v	v	v		v	v		v		
16	v	v	v				v			v	v		v		
17	v	v		v	v	v	v			v				v	
18	v		v				v		v	v					v
19	v	v	v	v		v	v		v	v	v		v		v
20	v	~					~		v	v	v		v		v
21		v			v		v			v	v		v		v
22	v	v	v			v	v		v	~				v	
23		v	v			v	v			v				v	~
24	v	v	v				v			v				v	
25	v	v					v		v	v	v		v		v
26		v		v			v	v		v					
27	v	v	v	v			v		v	v				v	

[27] Il est entendu par là : un outil/technique qui récupère l'entièreté de la variabilité et s'applicable hors du cas montré/testé.

[28] Il est entendu par là : un outil/technique qui permet la récupération d'une partie (plus ou moins grande) de la variabilité.

28	v	v	v			v	v		v	v				v	
29	v	v	v		v		v		v	v	v		v		v
30*							v				v		v		v
31															
32	v	v	v				v		v	v	v			v	
33	v	v		v	v		v			v				v	
34	v										v			v	v
35		v					v		v	v				v	
36	v	v		v	v		v			v	v		v		v
37	v	v	v		v		v		v	v	~		v		v
38	v	v			v	v	v			v	v			v	v
39	v	v		v	v		v		v	v				v	
40	v	v				v	v			v				v	
41*		v					v			v	v				
42	v	v					v			v					
43	v	v					v		v	v					
44	v	v					v			v	v		v		v
45		v				v	v			v				v	
46		v	v			v				v			v		v
47	v	v	v			v				v				v	
48	v	v	v	v			v	v		v			v		
49	v	v		v			v		v	v	v		v		v
50	v	v				v	v		v	v	v		v		v
51	v			~					v						
52	v		v	v	v				v	v				v	
53	v	v	v			v	v			v			v		
54	v	v		v	v		v			v	v		v		v
55	v		v	v	v	v	v			v			v		
56		v	v				v			v				v	
57	v		v	v	v	v	v		v	v			v		
58	v	v		v	v				v	v			v		
59*	v			v							v				
60	v	v	v	v		v	v			v	v	v	v		
61	v	v		v			v			v			v		v
62	v	v		v	v		v			v	v		v		v
63	v									v	v		v		
64	v	v	v	v		v	v			v			v		v
65	v	v	v	v		v	v		v	v	v		v		v
66	v														
67	v	v	v				v			v			v		
68		v		v		v	v			v	v		v		
69	v		v	v	v	v	v		v		v		v		v
70*		v					v								
71		v			v	v	v		v	v				v	v
72	v	v		v			v			v	v		v		

* = basé sur l'abstract et/ou l'introduction

4 Phase 4 – analyses et interprétations des résultats

Les différentes facettes décrites précédemment reprennent déjà les documents dans lesquelles elles apparaissent. Dans cette section, il ne s'agira d'interpréter cette mapping. A cette fin, chaque question de recherche est passée en revue et des diagrammes sont proposés pour exprimer visuellement les réponses déductibles (certaines questions de recherches se couplant).

4.1 Résultats et analyse de la mapping study

➢ Les techniques connues

Lors des différentes phases d'analyses, certaines techniques ont été trouvées (voir ci-après) mais hélas, aucune de ces techniques n'est nommée. En fait, il est très régulier de n'avoir, dans les publications scientifiques, aucun nom sur les techniques proposées. C'est d'autant plus vrai dans les domaines en voie d'exploration ou les méthodes n'ont pu réellement être validées, acceptées et nommées. Ce constat s'est aussi observé dans la publication [26] (« Feature location in source code: A taxonomy and survey ») qui est une taxonomie et enquêtes sur les techniques de localisation de features depuis un code source (remarque : cet article ne prend pas en compte d'autres sources que les codes sources) et qui ne nomme aucune technique mais reprend uniquement les noms d'auteurs et références de la publication proposant la technique.

➢ Les sources d'entrée et artéfacts de sortie

La Figure 33 de la page 88 représente, le nombre de publications trouvées en fonction de leurs sources d'entrée et type d'artéfact produit en sortie. Ce graphique est déduit des publications mentionnées dans les graphes des facettes précédentes (Figure 28 et Figure 29). Il faut remarquer que, par effet direct du principe de keywording orienté question de recherche, tout mot-clé de publication ne reflète pas nécessairement 1 méthode avec 1 source d'entrée et 1 artéfact en sortie. Il est donc possible qu'une publication reprenne plusieurs sources d'entrée et/ou reprenne plusieurs méthodes/approches (se basant elles-mêmes sur plusieurs sources d'entrée avec plus ou moins de sources de sorties suivant l'orientation des articles).
Par exemple, les publications [3], [7], [16] et [72] mentionnent plusieurs techniques ou sources d'entrée:

- [3] :
  Cette publication se base d'une part sur le Kbuild system (la propre structure de haut niveau du système dont ses features sont représentées grâces aux différents scripts présents dans le système et reprenant les features comme par exemple un listing des drivers présents dans le système, les autres étant absents et donc non obligatoire dans le système car celui-ci est considéré comme produit conforme aux spécifications) mais aussi, plus profondément, les codes sources structurés grâce aux indications « pre-processor symbols » de type « ifdef-blocks » que contiennent ceux-ci.

- [7] :
  Par la nature de cette publication et son but (recherche d'évaluation de techniques de rétro-ingénierie dans le but d'obtenir un ou des FMs), il est tout à fait normal de trouver plusieurs techniques dans celle-ci ainsi que plusieurs sources d'entrée pour un artéfact de sortie unique. Cette publication renseigne plusieurs techniques prenant des sources d'entrée comme : « configuration scripts », « propositional logic axpression », « natural language » ...

- [16] :
  L'abstract de cette publication mentionne clairement une synthèse de FMs depuis des « propositional constraintes ».
  Le contenu, quant à lui, reprend aussi des techniques prenant différentes entrées comme « product configuration », « feature dependences » ainsi que « code source ».

- [72] :
  Cette publication reprend un environnement (WebFML) permettant la synthétisation de FMs depuis plusieurs sources d'entrée. Ces sources sont : « product comparison matrices », « dependency graphs », « compilation directives and sources code », « propositional formulas », « a (set of) FM(s) ».
  Il en résulte donc plusieurs sources d'entrée pour un unique artéfact de sortie.

Le principe pour la mapping study étant de découvrir, sur base des publications, les directions déjà explorées, une suite possible que voient les industriels serait de donner ces directions aux praticiens (Petersen, et al. s.d.), afin qu'ils puissent sélectionner les publications qu'ils choisiront pour effectuer une « systematic literature review » en fonction de leurs buts (par exemple, s'ils se trouvent devant un code source O.O tel que Java et qu'ils désirent extraire la variabilité sous forme de FM, il leur suffira de reprendre la liste des publications ayant comme source d'entrée « code source O.O » et artéfact de sortie « feature models ») ou encore, dans le cadre de la recherche, de s'orienter dans les directions non/pas assez encore explorées.

L'utilisation de graphes à bulles permet explicitement et rapidement de mettre en évidence les orientations les plus exploitées mais aussi celles en manque d'exploration. De plus, ces graphes permettent de coupler des données (axe x et y). Un 3eme axe Z aurait pu être proposé. Cet axe aurait été, par exemple, la RIV depuis l'analyse logicielle ou depuis son implémentation. Néanmoins, pour plus de clarté et étant donné le peu d'information que cela aurait apporté, il a été choisi de travailler en 2 dimensions.

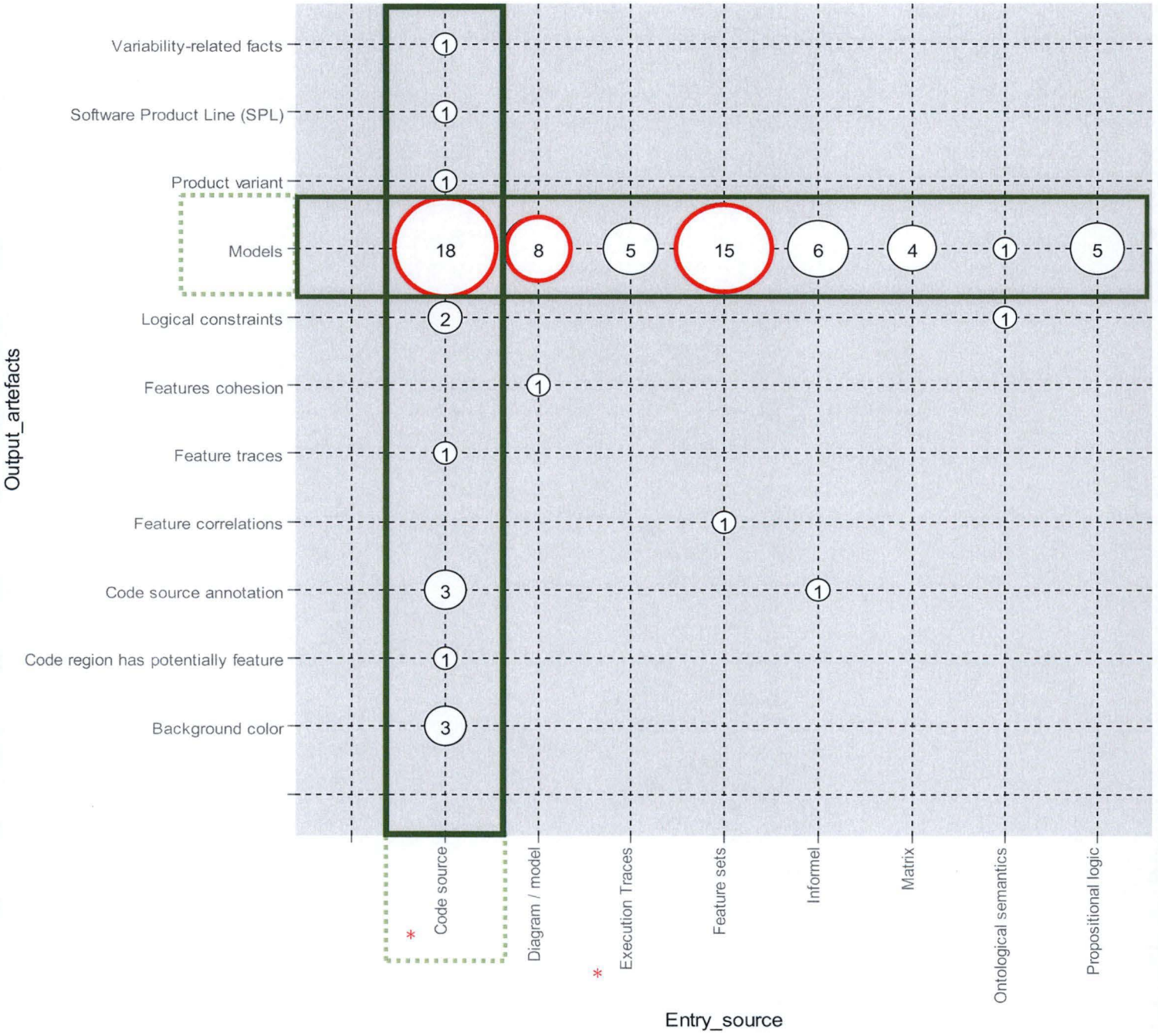


Figure 33 - Graphique des quantités de documents trouvés en fonction de leurs sources d'entrée et artéfacts de sortie

* = provenant de l'implémentation

Une première remarque constatée est que la RIV peut s'effectuer depuis 2 niveaux : depuis l'implémentation (code source et traces d'exécution), comme on pouvait légitimement s'en douter, mais aussi depuis l'analyse du système (ex. : depuis des descriptions de features, des informations informelles, des matrices ... ou encore depuis d'autres modèles).

Par exemple, la publication [6] propose une nouvelle approche afin d'extraire un *FM depuis un code source O.O.* L'approche va dans un premier temps extraire la structure (packages, classes, méthodes, attributs ...) depuis ce code source. L'étape suivante consiste à extraire les variabilités et commonalities en se basant sur l'analyse de type « formal concept analysis ». L'approche propose ensuite l'extraction des features par le biais de l'indexation « latent semantic indexing » (LSI)[29]. Ces features étant maintenant connues, un outil de modélisation suffit pour construire le FM. Néanmoins, cette approche n'est pas complète, les auteurs expriment très clairement le manque de contraintes. Ils laissent donc cette étape pour un futur travail.

Une autre approche permettant cette fois-ci d'extraire un *FM depuis l'analyse d'une application* est présentée dans l'article [56]. Cette nouvelle approche propose de partir d'un ensemble de produits (n'étant pas tous les produits possibles de la LPL). Partant de cet ensemble, la technique propose d'identifier les attributs des produits et donc, les features (sur base de l'analyse de type « formal concept analysis » (FCA)). Vient ensuite une phase d'extraction de la racine parente du FM et features obligatoires. Les CTCs de type « And » sont ensuite ajoutées et suivies des CTCs « Xor » selon des algorithmes précis. Les contraintes « Or », « inclusions » et « exclusions » sont par la suite respectivement ajoutées au FM. A la fin de cette technique, un FM complet est donc formé sur base de l'analyse logicielle.

Ce constat rejoint aussi la définition de la rétro-ingénierie (Chapitre II.2.2) qui, pour rappel, mentionnait le fait qu'elle permettait l'expression du système sous un niveau d'abstraction identique (ex. : de modèle à modèle) ou plus élevé (ex. : de code source à modèle).

Ce graphique permet aussi de faire directement ressortir 4 grandes conclusions :

- 39,24 % des méthodes sur base d'artéfacts provenant directement du code source et 78,48 % des artéfacts de sortie sont des modèles. Une hypothèse serait de partir du fait que la rétro-ingénierie permet l'extraction d'artéfacts permettant d'améliorer la compréhension d'un système. En partant du principe qu'un développement « standard » d'un système commence par une modélisation de ce système, il est légitime de penser que le retour en arrière permet donc l'extraction de modèles.

- 51,89 % des publications décrites au premier point reprennent des techniques exploitant les couples « artéfacts de source entrée/artéfacts de sortie » suivant :
  - « Code source / Models » (43,9 % de ces publications).
  - « Feature sets/ Models » (19,5 % de ces publications).
  - « Diagram / Models » (36,58 % de ces publications).

[29] Cette technique basée sur des méthodes statistiques, permet d'analyser et d'identifier les relations entre une requête formulée par un utilisateur (ex. : enregistrement automatique de fichiers) et un ensemble de documents contenant du texte. Un exemple complet et détaillé se trouve dans la publication de Rubin et Chechik (Rubin et Chechik 2013).

- Une tendance horizontale est présente en termes de quantité de documents au niveau de la facette « Models » (78,48 %).
  Cette tendance vient confirmer la théorie du chapitre Chapitre II qui mentionne le fait que, dans la littérature, la variabilité est régulièrement représentée sous la forme de modèle (et plus précisément sous la forme de FM).

- Une tendance verticale est présente en termes de quantité de documents au niveau de la facette « Code source » (39,24 %). Ce constat peut être interprété comme une tendance générale à partir du niveau le plus bas d'un système pour en déduire des artéfacts de sortie les plus représentatifs de l'implémentation.

Dès lors, il est intéressant d'explorer ces voies et d'affiner les recherches. Pour ce, il faut revenir sur les facettes les plus importantes et considérer chaque facette particulière comme étant une nouvelle racine d'un propre arbre à facettes. Il en ressort ainsi un graphe à bulles bien plus précis et ciblé sur les facettes voulues.

La Figure 34 ci-après affine le tri sur les 18 documents du couple « Code source / Models ».
En analysant ces résultats, il apparait très clairement que les modèles de type « feature models » sont les plus repris dans ce domaine (88.23 %).

La Figure 35 reprend les détails concernant les artéfacts de sortie possibles suivant les sources d'entrée de la facette « Feature sets ». Encore une fois, il en ressort que la rétro-ingénierie vers les feature models est une voie privilégiée par les scientifiques.

En analysant de plus près l'ensemble des résultats, il en ressort une tendance claire et nette à converger vers l'utilisation des FMs pour l'expression de la variabilité et ce, depuis différentes sources d'entrée se basant sur différents artéfacts intermédiaires.
A contrario, les voies menant vers d'autres types de structures finales comme des structures comportementales (FTSs ou autre) pouvant aussi permettre l'expression de la variabilité (chapitre Chapitre II.3.3 p.43) ne sont pas prisées.

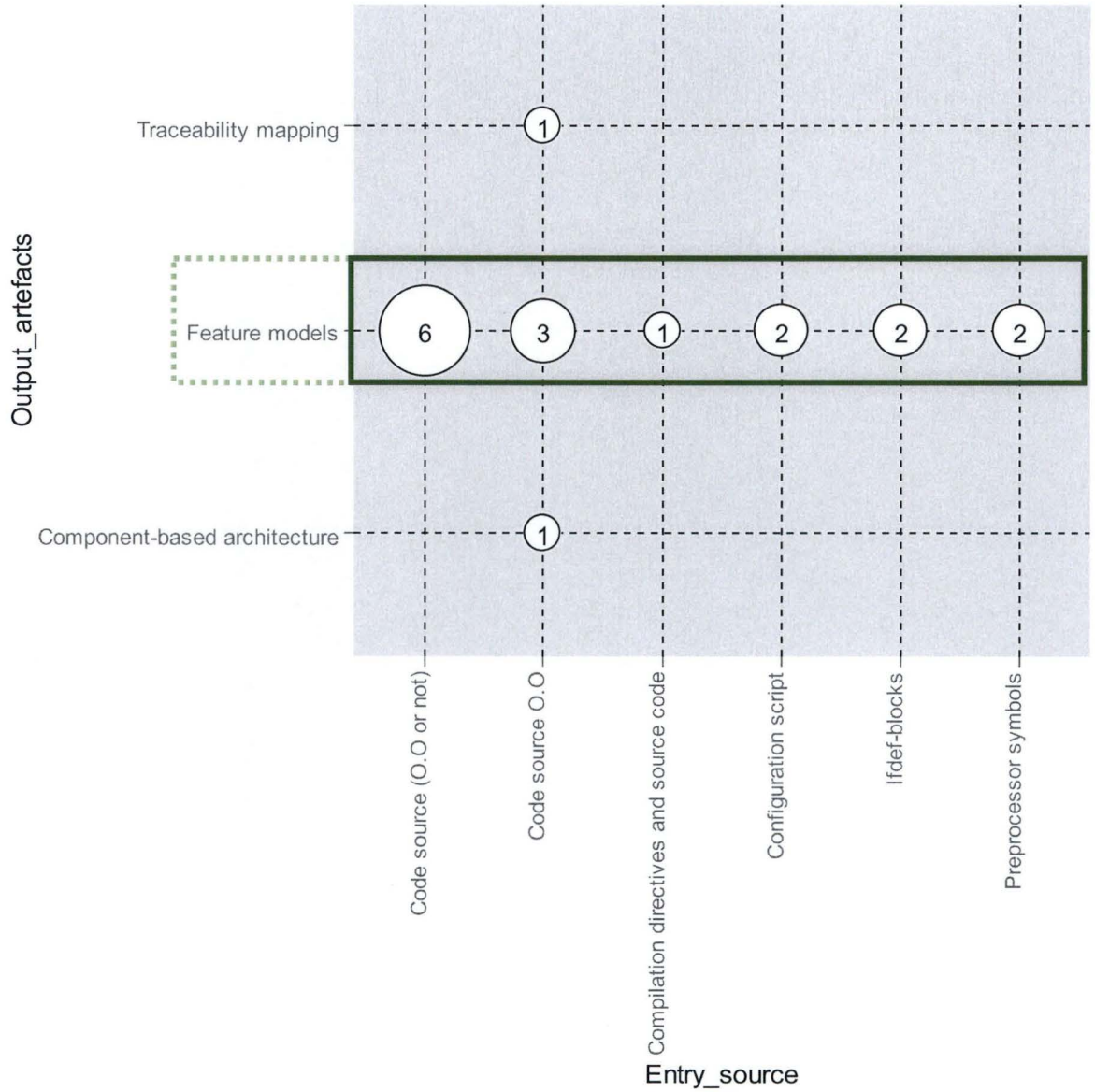


Figure 34 - Graphique des sources d'entrée et artéfacts de sortie des techniques repérées dans les publications focalisées sur le couple « code source / models »

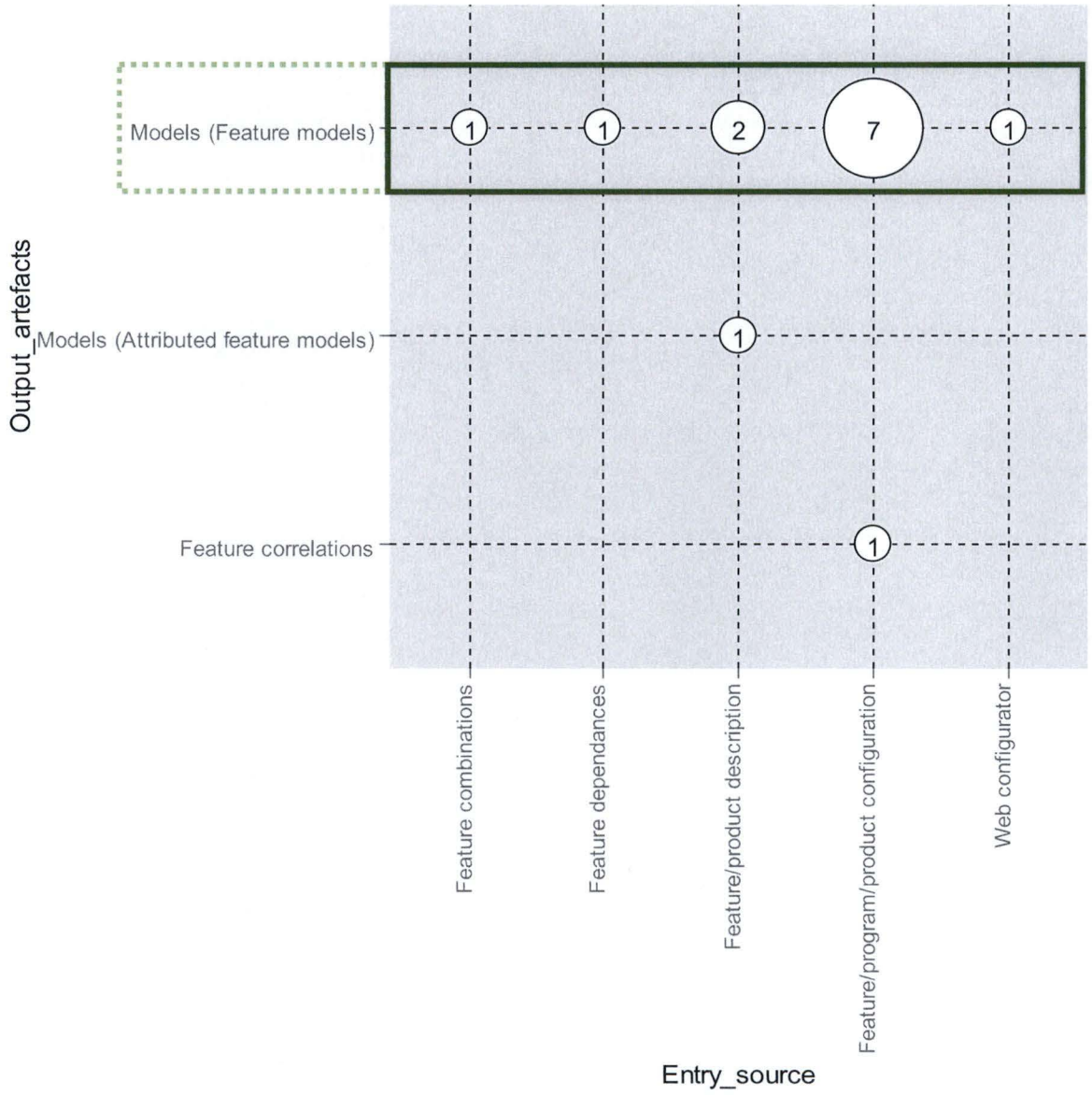


Figure 35 - Graphique des sources d'entrée et artéfacts de sortie des techniques repérées dans les publications focalisées sur la source d'entrée « Feature sets »

➢ Les mécanismes de gestion/expression de la variabilité

Il existe différents moyens d'exprimer et gérer la variabilité d'un système dont certains ont été décrits précédemment (Chapitre II.1.4.1). Parmi ceux-ci, certains sont repris dans le cadre des approches de rétro-ingénierie. Le graphique de la Figure 36 ci-après reprend l'ensemble de ces mécanismes en mentionnant le nombre d'articles dans lesquels ces mécanismes sont cités.

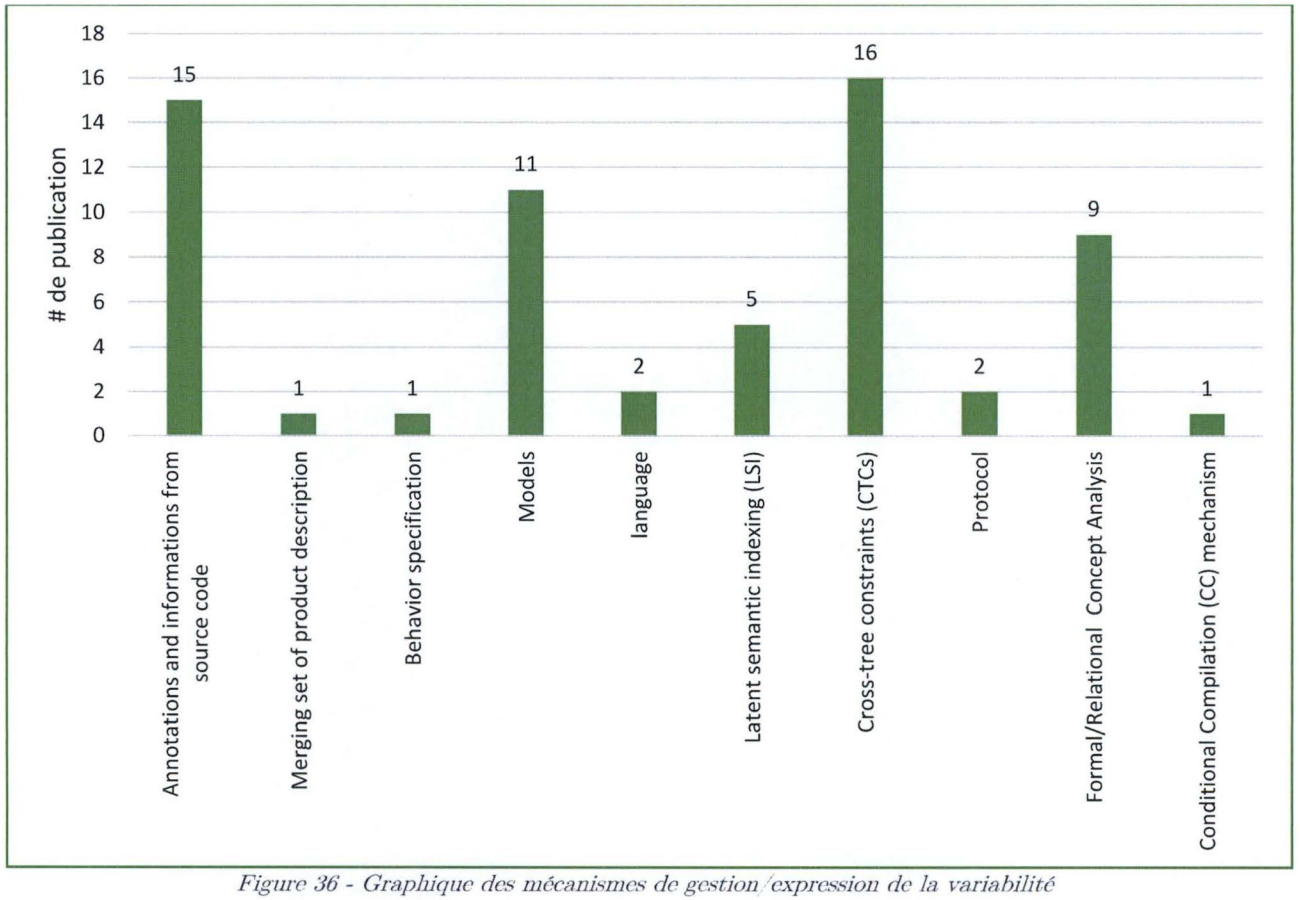


Figure 36 - Graphique des mécanismes de gestion/expression de la variabilité

Concernant les annotations provenant du code source, 46 % d'entre elles sont en fait les préprocesseurs de type « ifdef-blocks ».
Un outil d'ailleurs souvent associé à l'expressivité de la variabilité par annotation est « FeatureCommander ». Cet outil se base sur les informations telles que « ifdef-blocks ».

Un exemple concret de cette expression de variabilité est donné et comparé à sa forme de FM dans la publication [44] et repris dans la Figure 37 ci-après.

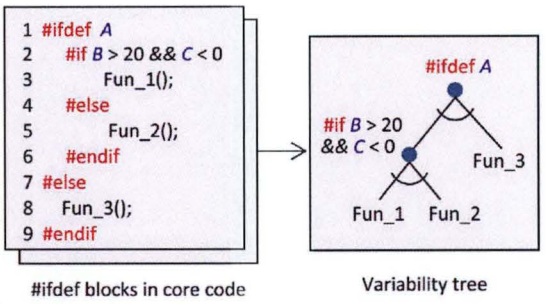


Figure 37 – Expression de la variabilité par interprétation de ifdef-blocks (Zhang et Becker 2013)

Une seconde remarque peut être formulée : les contraintes « cross-tree constraints » sont des contraintes (AND, OR, Xor) pouvant être appliquées aux feature diagrams et exprimées grâce à l'utilisation de logique propositionnelle pour ainsi donner naissance à un FM. Elles permettent donc *d'exprimer* visuellement la variabilité.

En analysant ce graphique, il apparaît clairement que les CTCs sont les moyens les plus utilisés pour gérer et exprimer la variabilité au sein des méthodes de rétro-ingénierie. En fait, étant une représentation directe des formules propositionnelles (Chapitre II.1.4.4), elles ont l'énorme avantage de pouvoir être mathématiquement calculées (et donc informatiquement gérées aussi) mais également, être graphiquement représentées. Elles sont donc parfaites pour ce genre de méthode qui calcule puis génère souvent un FM.

Les analyses de type « formal concept analysis » et « relational concept analysis » sont aussi des moyens souvent utilisés pour gérer la variabilité (voir ci-après). En fait, ce sont des analyses permettant l'expression de la variabilité de manière formelle (la syntaxe et la sémantique sont finies et exactes). Il est donc dès lors possible de les manipuler totalement et formellement. C'est pour cette raison qu'elles ont leurs places dans une facette particulière. Il est d'ailleurs à constater que ces analyses sont souvent utilisées dans les méthodes de RIV et ce, en prenant une place importante dans la méthode. Par exemple, la publication [56] détaille une méthode (voir ci-avant) permettant la rétro-ingénierie de FM depuis des configurations en se basant essentiellement sur l'analyse FCA car elle permet une expression complète et formelle de la variabilité permettant ainsi de lui appliquer des algorithmes concrets.

Après une analyse plus globale de ces informations, il en dégage un manque clair de moyens d'exprimer la variabilité par le biais de langage textuel exclusivement créé à cet effet. Il existe en effet des moyens d'exprimer des modèles de variabilité dans les différents outils récupérés mais aucun n'est un langage à part entière permettant d'exprimer exclusivement la variabilité de manière textuelle (à l'exception du langage TVL) et de manière complètement informelle (car exprimés en langue informelle) tels que par le biais de use-cases UML.

➢ Les mécanismes intermédiaires

Le graphique de la Figure 38 ci-après, représente tous les mécanismes intermédiaires utilisés par les méthodes de RIV. Pour rappel, ce sont les mécanismes (méthodes, modèles, graphes, langages ...) qui sont utilisés pour une ou plusieurs étapes au sein des techniques de RIV sans être la source d'entrée ni artéfact de sortie. Il y a 3 grands constats sur lesquels il vaut la peine de s'attarder.

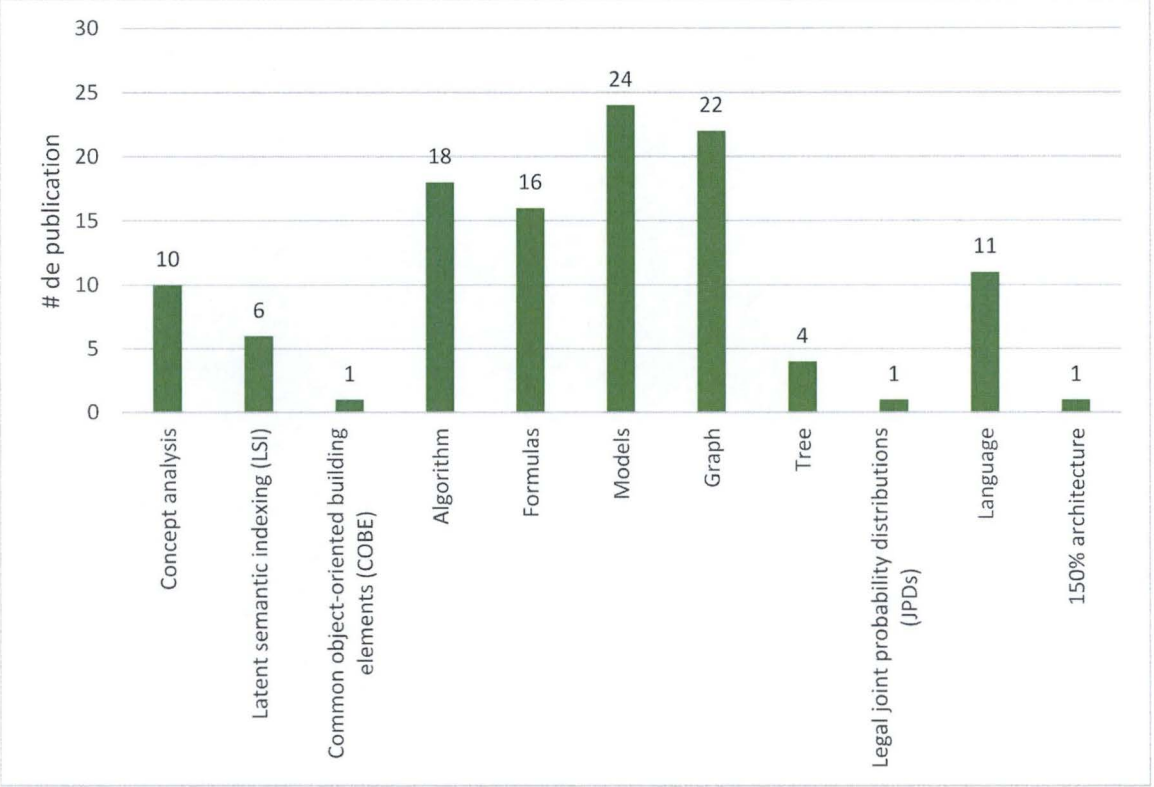


Figure 38 - Graphique des facettes des mécanismes intermédiaires

- Un premier constat se rapporte à de la facette « concept analysis ». Ce mécanisme se décompose en 2 sous mécanismes « Formal Concept Analysis (FCA) » et « Relational Concept Analysis (RCA) ». 90 % des publications contenant l'un ou l'autre mécanisme relatent, la FCA. Seule la publication [12] relate la RCA qui est, une variante de la FCA.
  Cela vaut la peine d'expliquer en quoi la RCA est une variante. Pour ce, il faut tout d'abord expliquer sa base, la FCA :

  Pour ce faire, l'article de Tilley et al. (Tilley, et al. s.d.) introduit la FCA dans le domaine de l'ingénierie logiciel :

> « Dans le domaine de l'ingénierie logiciel, la **Formal Concept Analysis** est typiquement appliqué pour les activités de maintenances, la reconstruction ou la modification de code existant et pour identifier les structures orientées objet (OO) »
>
> (traduit par PATINY Mathieu)

La FCA est une analyse qui consiste à formaliser la notion de concept (Xavier Dolques 2011). Un concept, est par exemple, la notion de « objet de sport ». Une multitude de caractéristiques lui sont attribuées, citons ainsi : ballon, volant, chaussure, jambière, crosse, gants, casque ... Ce concept est organisé de manière hiérarchique : un ballon de football est un ballon, un ballon de basket est un ballon, un ballon de rugby est un ballon, une crosse du sport lacrosse est une crosse, une crosse de hockey est une crosse, un volant de badminton est un volant, un volant de voiture de course est un volant ... Comme dans toute structure d'héritage, les attributs parents sont hérités par les enfants.
Un contexte formel (toute chose que l'on sait sur le « monde ») peut être vu sous forme de tableau :

Tableau 9 - Exemple de contextes formels dans une FCA

	Rond	Roule	Se porte	Solide	En cuir
Ballon	v	v			v
Volant	v			v	v
Chaussure			v		v
Jambière			v	v	
Crosse				v	
Gant			v		v
Casque		v	v	v	

Les concepts sont dérivables directement et représentés sous la forme de pairs (X, Y) où X est l'ensemble des attributs communs à un objet et Y, l'ensemble des objets qui ont ces attributs.

Dans le monde logiciel, cette analyse est régulièrement appliquée sur les anciens systèmes (ex. : la migration de programmes impératifs vers le paradigme orienté objet, la détection de paterns ...). Pour approfondir le domaine, l'article de Dolques et al. propose bon nombre de références en la matière.

La RCA est une extension de la FCA dont la différence se trouve dans l'ajout de relation autre que hiérarchique entre les objets (ex. : Une crosse se tient avec un gant en cuir et requiert le port de jambières), attributs ou concepts. Cette extension ouvre les portes vers une gestion des concepts beaucoup plus complexe. Pour approfondir les connaissances en la matière, la dissertation de Prieß (Prieß 1996) propose d'analyser de plus près cette extension.

- Un second constat concerne la facette « Algorithm ». Cette facette renferme 1 algorithme « Knowledge synthesis specification based (KSS) » (algorithme basé sur les KSS qui sont des spécifications d'un FM), 1 algorithme de « code cloning » (algorithme dont le but est de trouver les features implémentées par « clone and own »[30]) et 1 algorithme nommé « ETHOM » (algorithme évolutionnaire

[30] Principe qui consiste à copier une partie plus ou moins grande de code source proche du code à développer et de l'adapter (Krueger 2009), provocant par la même occasion, une scission entre les produits. Principe allant à l'encontre des systèmes développés sous la forme de ligne de produits logiciel.

permettant l'encodage de FMs sous la même forme que le chromosome). Les 13 autres algorithmes intermédiaires utilisés dans les publications sont des algorithmes de type heuristique. Ce type d'algorithme est utilisé dans le cadre de l'obtention d'une solution approchée grâce à l'expérience et les analogies[31]. Il est donc légitime de conclure que pour la majorité des publications récupérées, il n'existe pas de solution optimale. Cette conclusion peut amener à se poser la question suivante : existe-t-il des méthodes pour évaluer la couverture réelle d'un algorithme sachant qu'il n'existe que des algorithmes qui permettent de s'en rapprocher ? Une réponse sera apportée par la suite.

- Un 3ème constat est à remarquer sur la facette « language ». Il s'avère qu'il existe plusieurs langages[32] de modélisation sur lesquels les méthodes s'appuient dont un sort du lot par le nombre de publications le référençant : « Familiar ». Ce langage étant aussi repris comme un outil permettant d'effectuer de multiples manipulations sur les FMs (il est détaillé par la suite dans la section appropriée). Néanmoins il est utile de remarquer qu'il est un incontournable dans les langages de modélisation de FMs dans le but de la rétro-ingénierie.

En analysant de manière plus globale ces facettes, il en ressort qu'il existe beaucoups d'artéfacts intermédiaires sur lesquels les méthodes se basent. Ils sont riches et très variés. Constat qui pourrait être interprété comme « domaine instable et dont les chercheurs essaient différentes approches basées sur de multiples artéfacts différents ».

Il est aussi intéressant de constater que les structures sous forme d'arbres (graphes) ne sont réellement exploitées que dans 4 articles (dont, par exemple, l'article [4] utilisant des structures de type « abstract syntax tree » propose une technique semi-automatique permettant la RIV de 50 % à 100 % de précision). Les arbres sont donc des artéfacts dont la litterature est peu fournie.

[31] J.-F. Scheid « Chapitre 9 : Introduction aux méthodes heuristiques », TELECOM Nancy (http://iecl.univ-lorraine.fr/~Jean-Francois.Scheid/Enseignement/heuristiques.pdf)

[32] Ces langages ne sont pas spécialement orientés expression de la variabilité. Ils sont d'ailleurs bien plus souvent des langages créés pour les outils et permettent essentiellement une représentation de modèles sous la forme de texte plus qu'autre chose (sérialisation d'un FM).

➢ Les outils

Afin d'avoir une idée générale sur l'utilité de chaque outil, voici un référentiel de ceux-ci avec une brève description :

❖ Adora tool (http://www.ifi.uzh.ch/rerg/research/adora/tool.html)
Bien qu'aussi un langage, Adora est un outil sous la forme de plugin de l'IDE « Eclipse ». Il se veut être un outil permettant la modélisation de la variabilité grâce à ses 60 % d'éléments graphiques en plus par rapport au langage de modélisation unifié « UML ».

❖ AHEAD tool suite (http://www.cs.utexas.edu/users/schwartz/ATS.html)[33]
Acronyme de « (**A**lgebraic **H**ierarchical **E**quations for **A**pplication **D**esign) est une suite d'outils (fonctionnant sous Java) de modélisation architecturale ayant une orientation FOP (feature oriented programming).

❖ BeTTy Framework (http://www.isa.us.es/betty/)
Ce Framework fonctionnant en Java est un outil (distribué sous le format jar) basé sur certains composants de base de l'outil FAMA. Il permet la génération automatique de FMs, l'automatisation de la génération de données pour effectuer des tests fonctionnels sur les FMs ... Il intègre aussi un certain nombre de composants facilitant la comparaison d'outils d'analyse de FMs.

❖ CCVisu (http://ccvisu.sosy-lab.org/)
Cet outil est un outil permettant la création de graphes orientés. Il se base sur les fichiers aux formats RSF (Relational Standard Format) et propose en sortie, soit un graphe sous son propre layout, soit un graphe en image tels que SVG, ou encore VRML. Associé à l'outil « fact-extractor », il permet ainsi l'analyse de la structure d'un logiciel.

❖ CIDE (http://wwwiti.cs.uni-magdeburg.de/iti_db/research/cide)
Acronyme de « **C**olored **I**ntegrated **D**evelopment **E**nvironment », CIDE est un outil de développement de logiciels provenant d'une LPL. Il est proposé sous la forme de plugin de l'IDE « Eclipse ». Il permet, en partant du code source et d'un FM, de mettre en couleurs les différentes features représentées dans ce code source. Il propose aussi une mise en forme des endroits colorés sous forme d'arbre appelé « ASTView ». Il propose aussi une fonctionnalité permettant la détection d'erreurs de typage dans l'entièreté de la LPL.

❖ ETHOM (Segura, et al. 2014)
Cet algorithme peut être compris comme un outil permettant la génération de FMs. En lui donnant un outil et une analyse à effectuer, il génère un FM d'une taille prédéfinie afin de maximiser l'aspect tel que le temps d'exécution et la consommation de mémoire de l'outil. Il peut ainsi, par exemple, se coupler avec l'outil BeTTy. Le but étant d'informer les développeurs des performances de l'outil fourni.

[33] Un tutoriel complet est fourni à l'adresse :
http://www.cs.utexas.edu/users/schwartz/ATS/fopdocs/AHEAD-Tutorial.pdf

- FAMA tools suite (http://www.isa.us.es/fama)
  Cette suite d'outils est une référence en la matière. Elle est composée d'un Framework et d'une suite permettant d'effectuer différents tests sur les FMs. Son but est d'automatiser les analyses faisables sur les FMs. Pour ce, elle implémente des outils tels que : BDD, SAT et CSP.
  Elle est composée d'une base sur laquelle il est possible d'ajouter une multitude d'extensions par-dessus (ex. : FaMa Feature Model, Sat4jReasoner, JavaBDDReasoner, ChocoReasoner ...) et a l'avantage d'être sous licence GNU et soutenuz par la Commission Européenne.

- Familiar (http://familiar-project.github.io)
  Acronyme de « FeAture Model scrIpt Language for manIpulation and Automatic Reasoning » est un outil sous la forme de plugin de l'IDE « Eclipse » (intégré dans le plugin « Feature IDE ») ou d'application standalone Java (en ligne de commandes ou dans un environnement complet[34]) créé dans le laboratoire I3S par Mathieu Acher, Philippe Collet et Philippe Lahire. Il est actuellement géré par l'équipe « Triskell » (composé de INRIA, IRISA et l'université de Rennes 1) ainsi que l'équipe « MODALIS » (I3S laboratoire et l'université de Nice Sophia Antipolis) et l'université « Colorado State University ». Il propose d'un côté, son propre langage permettant la modélisation de FMs et de l'autre, une foule de fonctions de manipulations de FMs (importer, exporter, décomposer, éditer, configurer et calculer des différences entre eux). Mentionné comme langage incontournable dans le processus de rétro-ingénierie, il est intéressant de le tester.
  Une série d'exemples proposés par le projet sont exécutés et interprétés en annexe VII.

  Un plus très avantageux : il supporte les modèles de type TVL (Text-based Variability Language) développés à l'université de Namur [35]. Ce langage permet la modélisation de FMs de façon la plus proche possible du langage plus qu'incontournable qu'est le C. Il a été conçu dans l'optique d'avoir une syntaxe C-like afin de faciliter son accès dans le monde des entreprises. En effet, un nouveau langage dont le temps d'apprentissage est drastiquement réduit grâce à la connaissance commune des développeurs est plus facilement adopté par les entreprises car il engendre un coût de formation minimum.
  Familiar se voit donc compatible avec un langage facilitant l'intégration en entreprise.
  Il est aussi compatible avec les formats : SPLOT et FeatureIDE.

- Feature Commander (http://www.infosun.fim.uni-passau.de/spl/janet/xenomai)
  Feature Commander est un logiciel prototype (portable et fonctionnant sous Microsoft Windows) permettant, en se basant sur les préprocesseurs d'un code source, d'assigner et de découvrir les différentes features. Ces features sont ensuite représentées visuellement par le biais de multiples couleurs configurables. Un arbre des features est ainsi proposé. L'annexe VIII montre 2 screenshots commentés permettant d'avoir une idée plus nette sur les fonctionnalités de ce logiciel.

---

[34] Téléchargeable à l'adresse : http://mathieuacher.com/pub/FAMILIAR/releases
[35] Pour en savoir plus : https://projects.info.unamur.be/tvl

- FeatureVisu (http://featurevisu.sosy-lab.org)
  Cet outil est une extension du logiciel « CCVisu »[36]. Il permet l'analyse et la mesure de LPLs. Il est utilisé dans le cadre de décompositions de graphes logiciels en sous-systèmes représentés par le biais de clusters (permettant ainsi l'amélioration de leur compréhension). Il est une extension dans le sens où il permet d'inclure dans la représentation, le principe de features. Il se base en entrée sur un graphe de LPL et son implémentation doit contenir des annotations comme préprocesseur ou encore sous la forme d'approches conditionnelles comme la programmation orientée feature[37].

- Fmp2rsm plugin (http://gsd.uwaterloo.ca/fmp2rsm)
  Sous la forme de plugin de l'IDE « Eclipse », cet outil (prototype) regroupe d'une part, un « Feature Modeling Plugin » et d'une autre part, « Rational Software Modeler ». Il est une extension permettant la modélisation de lignes de produits logiciels au sein d'UML ainsi que la dérivation automatique de produits correspondants.

- Framework RECoVar
  Acronyme de « Reverse Engineering Configurations and Variability", ce Framework incluts 2 approches : l'extraction de la variabilité basée sur les préprocesseurs présents dans le code source ainsi que l'approfondissement des corrélations entre features depuis les configurations de produits.

- Golem tool (https://www4.cs.fau.de/Research/VAMOS)
  L'outil Golem est une part du logiciel VAMOS (acronyme de « Variability Management in Operating Systems »). Cet outil permet l'analyse, basée sur les makefiles, de systèmes Linux et induit des contraintes de dépendances depuis les règles de build. Il est utilisable via des lignes de commande et implémenté en Python. L'article [3] utilise cet outil dans le but d'implémenter son algorithme.

- Javapp (http://slashdev.ca/javapp)
  Cet outil (développé en python et tournant sous Java par le biais de Jython[38]) répond à un besoin absent actuellement qu'est la génération de préprocesseurs dans les codes sources Java. Il répond à ce besoin en prenant en entrée un code source Java et de l'information (sous format XML) concernant les préprocesseurs à ajouter de manière intelligente.

- KBuildMiner
  Cet outil (développé via 1400 lignes de code source Scala et 450 en Java) est un parseur permettant, depuis un KBuild makefile, de retourner un abstract syntax tree (AST). Dans la publication [3], l'auteur a téléchargé et appliqué son approche sur Linux V2.6.33.3 . L'explication réside dans le fait que cette application a besoin d'une configuration lourde et manuelle de son makefile pour être adaptée à une version particulière de Linux.

---

[36] Pour en savoir plus : http://www.sosy-lab.org/~dbeyer/CCVisu
[37] Pour en savoir plus : http://en.wikipedia.org/wiki/Feature-oriented_programming
[38] http://www.jython.org

- LEADT tool
  Acronyme de « Location, Expansion, And Documentation Tool », LEADT est un prototype développé sur CIDE[39], permettant la localisation de features dans les codes sources Java. Il est proposé sous la forme de plugin de l'iDE « Eclipse » (V3.5 ou 3.6 avec Java 1.5).

- Orange tool (http://fosd.de/cide)
  Cet outil OpenSource est utilisé dans le cadre de la bio-informatique. L'article [44] l'utilise afin d'implémenter son algorithme dans le cadre de l'approfondissement de la corrélation entre features.

- REVPLINE
  Acronyme de « **RE**–engineering Software **V**ariants into Software **P**roduct **Line** » est un outil permettant de faire de la rétro-ingénierie de variabilité depuis un code source orienté objet dans le cadre d'un processus de re-engineering. Il a une approche particulière se basant sur les FCA, LSI et dépendances entre codes sources.
  Pour en savoir plus et approfondir cet outil, il est conseillé de lire la thèse de Al-Msie'Deen (Al-Msie'Deen 2014).

- WebFML[40]
  Cet outil, développé par l'aquipe Triskell (Inria / Irisa), est très étroitement lié avec l'outil « Familiar ». En effet, cette équipe fait aussi partie de l'équipe de recherche pour ce dernier outil. WebFML est un environnement de synthétisation de FMs depuis plusieurs sources d'entrée (ex. : comparaison de produits, des fichiers de configuration, des graphes de dépendances, des directives de compilation, des formules propositionnelles ou encore d'autres feature models). Son point fort réside dans le fait d'avoir un affichage visuel des feature models, de pouvoir exporter son résultat dans plusieurs formats mais surtout, d'être très interactif avec l'utilisateur permettant ainsi un suivi complet lors de la manipulation des FMs (ex. : sélection de parents candidats, proposition de regroupements de features, une gestion de retour en arrière, la génération de FMs en prenant compte d'heuristiques et des choix effectués ...).

[39] Virtual Separation of Concerns : http://fosd.de/cide
[40] Une présentation est disponible sur : https://github.com/FAMILIAR-project/familiar-documentation/tree/master/manual/webfml

En analysant les outils présentés, un constat s'impose : les outils partant du code source d'un système se basent bien souvent sur les préprocesseurs présents dans celui-ci. C'est donc une voie explorée qui, d'un premier abord, paraît être une direction proposant des solutions intéressantes.

La Figure 39 ci-après reprend le nombre de publications dans lesquelles ces outils sont utilisés ou mentionnés comme utilisés.

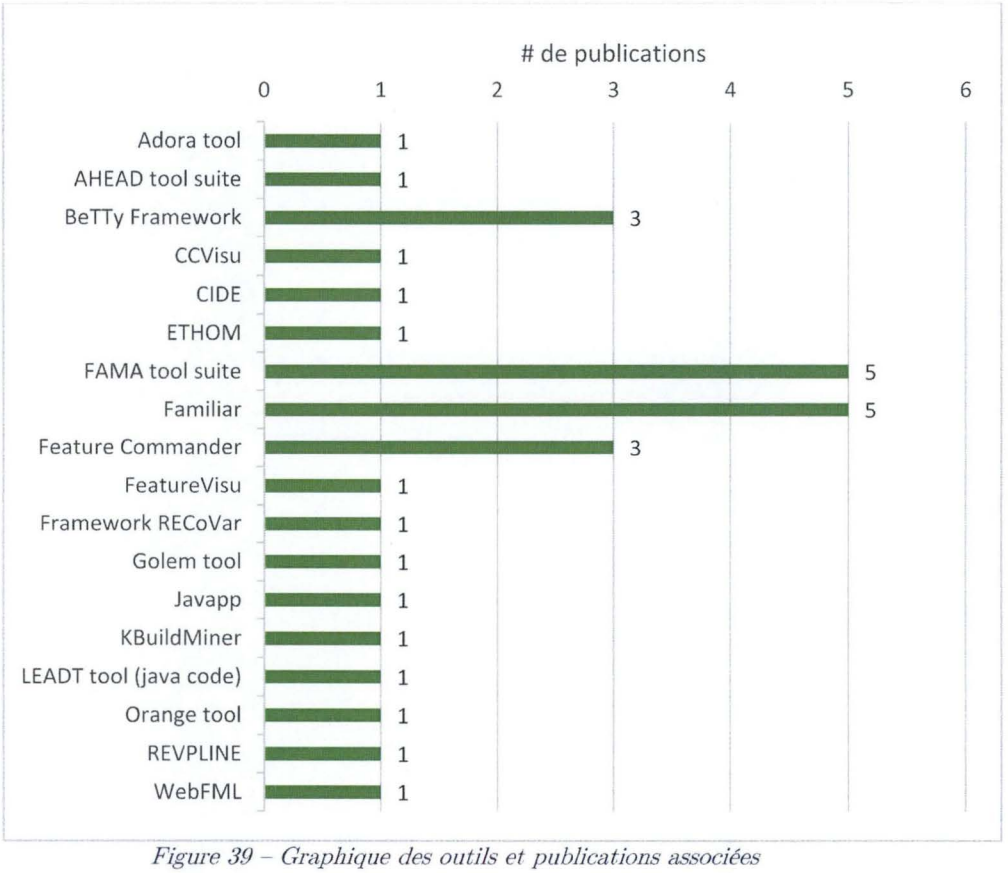


Figure 39 – Graphique des outils et publications associées

Lors de l'attribution des publications aux outils, 4 outils se démarquent par le nombre de publications associées. Ces outils sont « BeTTy Framework », « FAMA tool suite », « Familiar » et « Feature Commander » avec respectivement 3, 5, 5 et 3 publications. Par conséquent, ils peuvent être considérés comme des références en la matière dans les directions déjà explorées. Ceci peut s'expliquer par le nombre de fonctionnalités qu'ils proposent.

➢ Les types de publications et évolution dans le temps

Toutes les publications trouvées ont été préalablement classées suivant le type de contenu auquel elles se rapportent. Un graphe visuel permet d'avoir une idée rapide de leur classement.

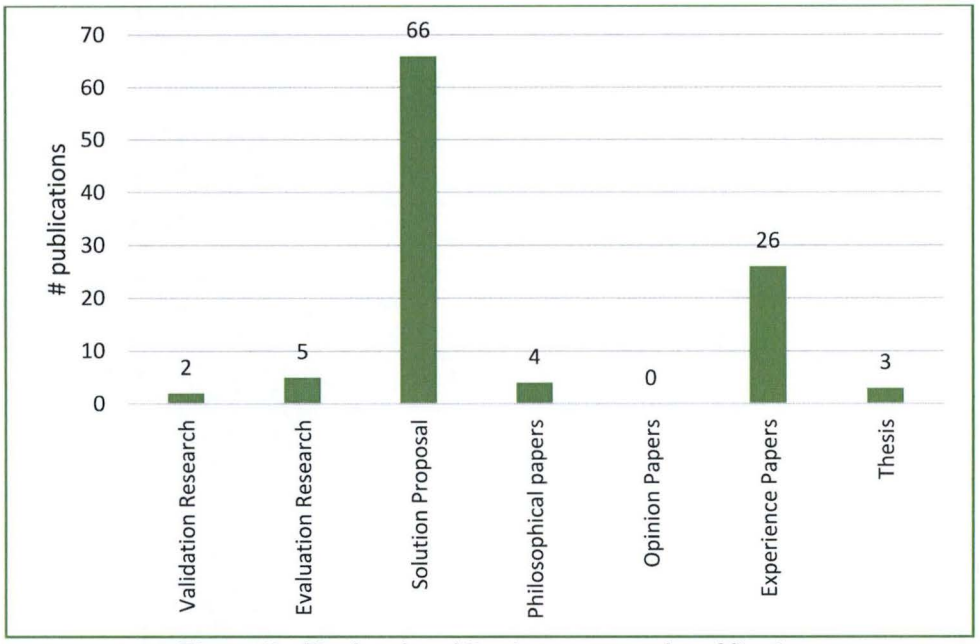


Figure 40 - Nombre de publications par type de publication

La majorité des publications trouvées sont très clairement des propositions de solutions (typiquement des méthodes et approches) et des rapports d'expériences (typiquement des études de cas). Le fait de n'avoir que très peu de validations et d'évaluations peut s'expliquer par le fait que ce domaine est encore en recherche de solutions et partage d'expériences plus qu'au perfectionnement et l'optimisation de solutions. Les résultats de cette mapping study viennent donc confirmer notre intuition de départ : la rétro-ingénierie de la variabilité dans les LPLs est peu mature.

Lors des recherches, aucune publication d'opinion personnelle n'a été trouvée. Cela résulte peut-être du fait qu'il ne se dégage pas vraiment de chercheurs ou groupes de chercheurs experts (et reconnus comme tels par la communauté scientifique) qui aient atteint un niveau de maturité (pour ce domaine précis uniquement) assez grand pour envisager d'exprimer des opinions et critiques tranchées sur les méthodes (peu nombreuses) proposées. Ou est-ce simplement que ce genre de publications à tendance à disparaitre au profit de publications à bases de preuves ...

Afin d'avoir une représentation sur le nombre de publications apparues au cours du temps, la Figure 41 ci-après propose une vision des types de publications en fonction du temps. Il est intéressant de remarquer que le plus grand nombre de publications trouvées sur le sujet date d'entre 2011 et 2014.
A noter que l'année 2015 étant seulement entamée lors des recherches, il est normal de ne trouver que peu de publications pour celle-ci.

Il apparaît aussi que le nombre de publications d'expériences suit la tendance du nombre de publications proposant une solution (suivant une proposions d'environ 50 % de ces derniers). Ce constat est normal dans la mesure où, souvent, les publications de solution mettent en œuvre, sous forme d'étude(s) de cas leur propre solution.

Il est donc encore une fois clairement perceptible que le domaine est en manque de validation des méthodes (que ça soit par des publications de validations, d'évaluations ou encore d'opinions). C'est donc une des tâches sur laquelle les chercheurs devront s'orienter pendant ces prochaines années.

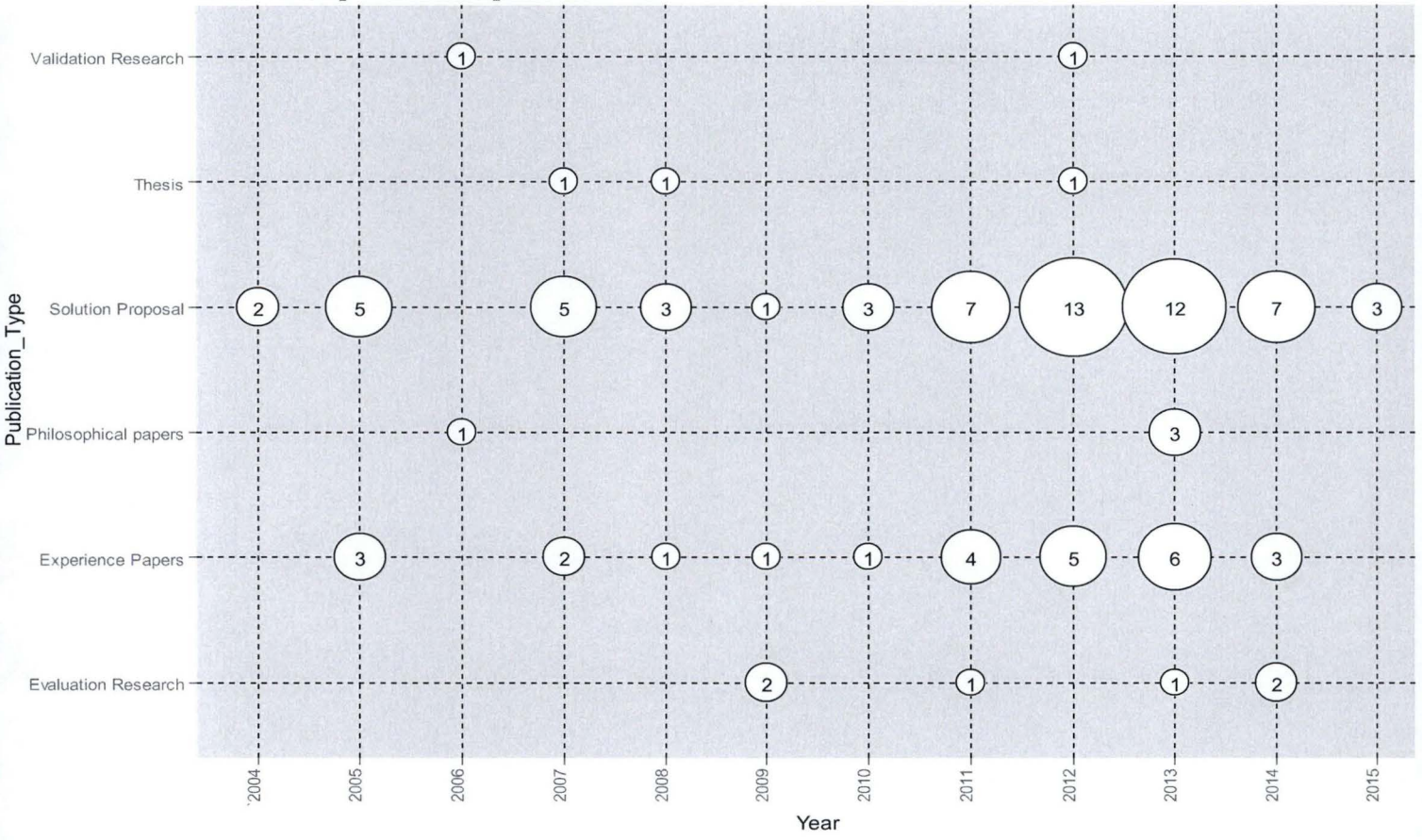


Figure 41 - Graphique du type de publication en fonction du temps

Une autre analyse pertinente à faire est de regarder, pour chaque couple de facettes de sources d'entrée/artéfacts de sortie récupérés, quels types de publications s'y sont intéressées. La Figure 42 ci-après exprime ces relations. Une attention toute particulière est mise sur 4 types de publications afin de mettre en évidence les couples qui seraient approchés, expérimentés, évalués et validés. Ce graphe permet d'exprimer visuellement le manque de publications pour certains couples (ex. : les techniques de RIV basées sur les traces d'exécutions sont seulement proposées théoriquement sans aucune expérimention, évaluation ni validation. Autre exemple : le couple feature sets/ models a déjà été approché dans 10 documents, 3 études de cas ont été menées, 3 études d'évaluations relatent ce couple, mais aucune ne valide cette approche.).

Ce graphe doit être considéré comme un état des lieux des types recherches menés dans le domaine et exprimant la maturité ou, comme dans ce cas, l'immaturité du domaine.

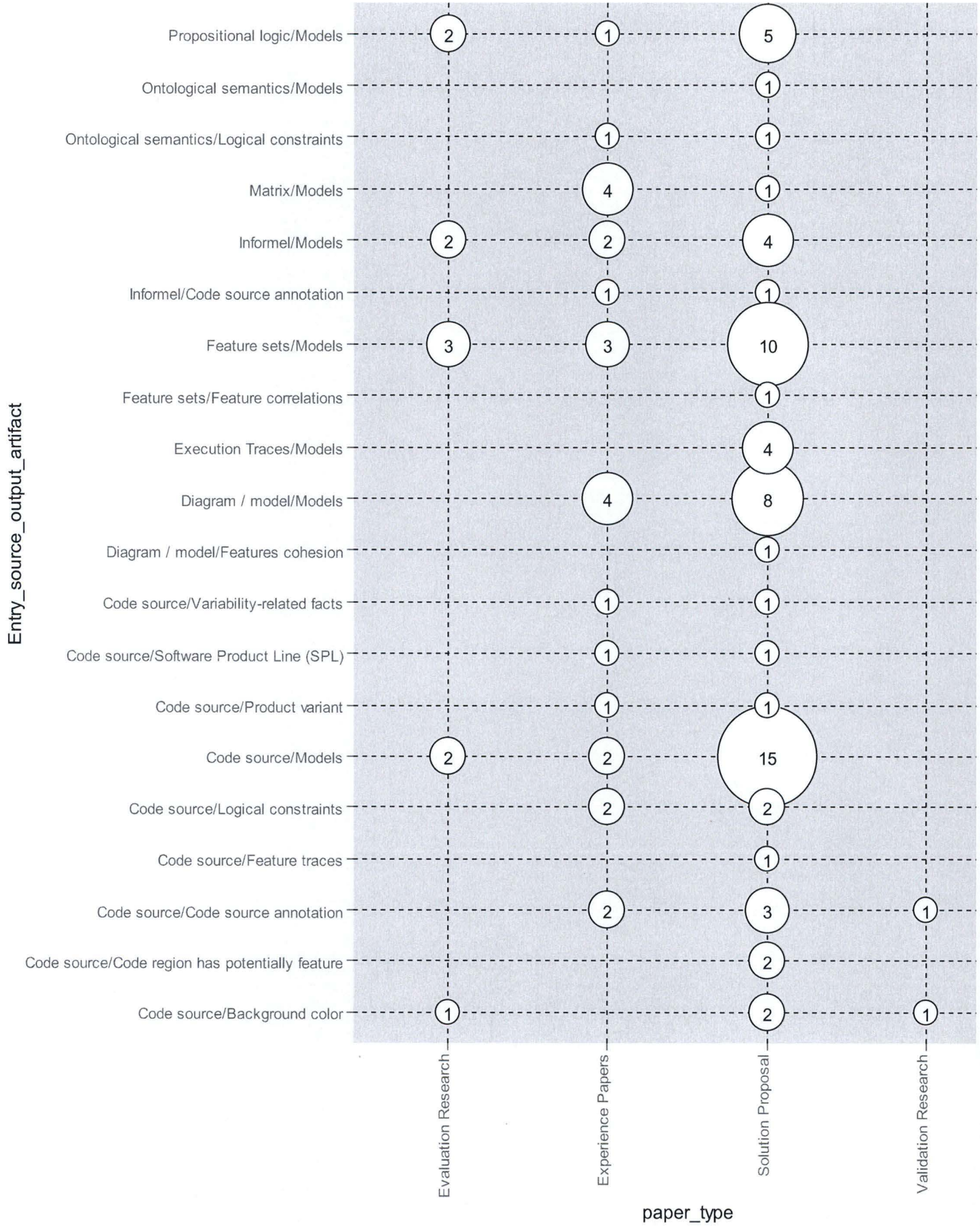


Figure 42 - Graphique des types de publications trouvées pour les couples de sources d'entrée/artéfacts de sortie

➢ Les techniques de recouvrement :

Les techniques permettant de donner une indication concernant le recouvrement d'une méthode/approche sont très peu nombreuses et informelles. Un besoin est pourtant présent afin de valider les techniques (donc beaucoup sont empiriques) de RIV.
Après réflexions, il est normal de n'en trouver que très peu dans le sens où les techniques de validation de modèles de variabilité font l'objet de leurs propres publications, et ce, sans nécessairement être orientées rétro-ingénierie. Par exemple, la publication de Henard et al. (Henard, et al. 2013) propose une technique dans laquelle le nombre de mauvaises contraintes entre features peut être récupéré. Cette technique pourrait être appliquée sur un processus de rétro-ingénierie dont l'artéfact de sortie est un FM. Le nombre d'erreurs donnerait une indication sur le pourcentage de recouvrement de la méthode.

A l'heure actuelle, la technique la plus utilisée pour vérifier une méthode de RIV est d'appliquer ces techniques sur des systèmes triviaux ou entièrement contrôlés. Cela permet, comme dans l'article [63], de mentionner les erreurs de découverte de features depuis un code source. Par exemple, cet article tente de valider l'hypothèse suivante : les pré-processors symboles peuvent servir à faciliter la compréhension d'un système en permettant l'annotation des features dans le code source d'un produit. Ce processus est entièrement manuel et le taux de validité des features découverte est effectué dans un environnement entièrement contrôlé.

Néanmoins, ce constat ouvre la porte sur de prochaines recherches ayant l'attention sur la couverture de recouvrement que propose un modèle (typiquement un FM), par rapport à un système implémenté.

5 Validité

Lorsqu'une systematic mapping study est menée, différents éléments peuvent influencer sa validité. Ces éléments sont appelés « threats to validity » en anglais et se traduit littéralement par « menaces de validité ». Les menaces concernant cette étude sont les suivantes :

- Choix des librairies en ligne et exhaustivité de la liste des publications
  Dès le début, une série de librairies en ligne ont été sélectionnées. Bien qu'il soit nécessaire de reprendre les librairies les plus connues pour ce domaine de recherche de ce travail, il est impossible de toutes les retenir. Il est donc possible que certaines librairies reprenant des articles pertinents pour ce mémoire, n'aient pas été sélectionnées. Pour contrer au mieux cette menace, il a été choisi de conserver les librairies conseillées par des chercheurs[41] (ex. : IEEE Xplore Digital Library, ScienceDirect ...) et ajouté un certain nombre de librairies auxiliaires orientées informatique (ex. : Directory of open access journals, FreeSearch Toward computer science ideas ...). En tout, ce n'est pas moins de 21 librairies en lignes qui ont été interrogées afin de rendre le plus exhaustif que possible la liste publications trouvées.

- Langue
  La langue peut être une barrière à la recherche de publications. En effet, certaines de ces dernières n'étant pas traduites dans la langue internationale de la recherche scientifique (l'anglais), il est possible qu'elles ne soient pas reprises, bien que pertinentes. L'anglais et le français ont été choisis afin d'élargir fortement, tout de même, le champ des recherches.

- Mots clés non connus du meneur
  Idéalement, le meneur de ce type de recherche devrait être accompagné d'un ou plusieurs experts du domaine afin de cibler au mieux les mots clés. En effet, certaines publications peuvent contenir des mots clés synonymes et/ou expressions inconnus du meneur. Dans le cas présent, les mots clés constituant la chaîne de recherche ont été validés par les co-promoteurs du travail. Cela dit, la suite constituée de filtres et multiples tris n'a pu être totalement revue dans les détails par les personnes accompagnant ce travail, car le cadre d'un mémoire ne permet pas d'effectuer une tâche entièrement évaluée par les pairs (entres autres, par manque de temps de ces derniers).

- Validité dans le temps
  Lorsqu'une recherche est menée, elle s'effectue à une date précise (ou pendant une courte période donnée). En soi, cela est plutôt un constat qu'une menace. Mais, cet aspect temporel doit être pris en compte dans l'interprétation des résultats. Cela est d'autant plus vrai pour les domaines en pleine expansion. Lors de l'analyse des différents résultats, il a été observé une augmentation du nombre d'articles jusqu'en février 2015 (1 mois avant la date de fin des recherches). Il est donc naturellement conséquent qu'une même recherche menée fin 2015 proposera de nouveaux résultats, et ce, en plus grande quantité et couvrant des nouvelles directions.

[41] Il s'agit des co-promoteurs de ce mémoire.

- Faux positifs
  Lorsqu'une recherche d'une telle envergure est menée par une seule personne (plus de 1500 publications à filtrer et trier pour en extraire 72), il se peut que certaines publications passent aux travers des filtres et se retrouvent incluses dans la liste des publications finales. Dans cette mapping study, après avoir pris du recul et effectué plusieurs lectures, il a été trouvé 2 faux positifs :

  - [14] (A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, and G. Saval, « Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis »)

  - [35] (K. Czarnecki and M. Antkiewicz, « Mapping features to models: A template approach based on superimposed variants »)

  De faux positifs peuvent typiquement apparaître lors des recherches approfondies, mais nullement dans la recherche principale (1ère recherche depuis les librairies en ligne). Cela remettrait en doute certains mots clés ou moteurs de recherche.
  C'est en effet le cas de ces 2 publications extraites de la littérature grise et relatant un mapping de features entre modèles de niveau d'abstraction différents. Ces publications auraient dû être éliminées lors de l'application des filtres suivant les critères prédéfinis, car elles ne relatent pas une technique de rétro-ingénierie menant à l'expression de la variabilité.

- Légère perte de précision lors de la création de facettes
  Lors de la phase 3 du protocole de cette mapping study, un processus permettant l'extraction de mots clés représentant chaque publication et leur fusion en facettes est appliqué. Il s'agit ni plus ni moins d'effectuer une abstraction (dont la définition de base mentionne le fait d'oublier les détails et de ne garder que l'essentiel voulu). Dans toute abstraction, il y a, inéluctablement, une perte d'information plus ou moins grande.
  Dans cette phase, aucune représentation particulière n'a été imposée concernant les démarches de génération et fusions des mots clés. Afin de proposer cette abstraction tout en gardant un maximum d'informations, il a été choisi de les représenter sous la forme d'arbres à nœuds colorés permettant ainsi, pour qui le souhaite, de récupérer les informations perdues au cours de cette abstraction. Néanmoins, la profondeur des arbres permet l'expression de mots clés très légèrement eux-mêmes abstraits.

Chapitre IV. Rétro-ingénierie de systèmes web

1 Synthèse des publications

Lors de la mapping study, il a été convenu d'élargir le champ de recherche en passant des systèmes web aux systèmes en général suite à un manque de publications exclusivement web. Ceci dit, lors de l'analyse et de l'interprétation des résultats, une remarque a émergé : la rétro-ingénierie peut se faire de l'implémentation d'un système (codes sources, traces d'exécutions ...) mais aussi de son analyse (modèles, descriptions de produits +/- formelles ...). Certaines des techniques précédentes basées sur cette analyse peuvent donc être appliquées à ce genre de systèmes même si ce n'est pas leurs buts originaux (tout en prenant compte de leurs contextes pas toujours totalement structurés (chapitre I p.11 à p.14)).

Afin de détailler ce qui a déjà été proposé dans ce domaine précis, la synthèse des 2 articles ressortis dans la mapping study exclusivement orienté web est reprise ci-après.

[b] S. Marciuska, C. Gencel, and P. Abrahamsson, "Automated feature identification in web applications," *Lecture Notes in Business Information Processing*, vol. 166 LNBIP, pp. 100–114, Nov. 2014.

Pour ce premier article, il s'agit de la présentation d'une approche et la mise en application via leur propre outil créé pour fonctionner avec leur algorithme. Le départ se fait non pas du code source du système comme beaucoup d'autres techniques (hors web) le proposent mais de l'interface utilisateur d'applications web (développé en HTML5). Pour ce, les auteurs sont partis de la définition suivante du mot « feature » :

> « Une **feature** est une réalisation fonctionnelle d'une exigence du système (ex. : une unité observable du comportement du système déclenché par l'utilisateur) »
> (traduit par PATINY Mathieu)

Partant de cette définition, ils l'ont étendue en ajoutant le fait que le déclencheur peut être humain mais aussi un autre système comme un web service. Cette extension leur a permis, dans leurs études de cas, de tester la différence entre un utilisateur humain découvrant les features et un outil automatique.

Un grand avantage des systèmes web est qu'ils sont analysables depuis leurs codes sources (pouvant être écrits, à l'origine, en plusieurs langages) qui se trouvent souvent complexes, mais surtout, depuis le navigateur web. Ce navigateur étant un point obligatoire de passage des données, il est aussi le mieux placé pour analyser l'entièreté de ces données. C'est donc sur cet artéfact intermédiaire qu'ils se sont basés.

Ils sont partis du principe que les systèmes web étaient proposés en HTML5, JavaScript et CSS (du côté client) et que chaque événement menait le système d'un état à un autre : un événement souris, un événement clavier ... ou encore un événement d'objet et formulaire (l'ensemble exhaustif de ces événements est repris en annexe IX).

Ils ajoutent 3 éléments statiques HTML5 permettant l'expression de features : « ancor », « input » (à l'exception du type hidden) et « textarea ».

Dans l'optique de retrouver ces éléments, un algorithme implémenté en JavaScript (langage côté client et pouvant exploiter toute la puissance du navigateur) parcourt le DOM (Document Object Model) afin de les trouver (cet algorithme est mis à disposition sur le site Internet : http://featurereduction.org). Au final, l'entièreté des actions possibles fournies par le système sont récupérées.

Plusieurs résultats d'études de cas sont proposés en fin d'article et permettent de démontrer, par l'expérimentation, que l'outil proposé est fonctionnel et permettent de comparer une découverte manuelle des features par rapport à l'outil automatique implémenté. Cette technique ouvre aussi la voie vers une nouvelle piste : l'identification de features au sein d'application RIA (rich Internet application)

[i] E. K. Abbasi, M. Acher, P. Heymans, and A. Cleve, "*Reverse engineering web configurators.*", 17th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, 2014.

Cette seconde publication consiste à retourner, par le biais d'une méthode semi-automatique, la variabilité présente dans des objets du monde matériel ou logiciel configurables grâce aux applications de type « web configurators ». Un exemple parmi des milliers d'autres pensables dans cet article est celui d'un configurateur de véhicules automobiles de la marque Audi.

La publication explique qu'une technique totalement automatique pour ce genre de systèmes n'est pas réaliste ni désirable dans le sens où la diversité des présentations et implémentations dans ce genre de configurateur est trop grande. Une supervision humaine est nécessaire.

Les auteurs mettent en avant 2 composants majeurs dans leur technique « web wrapper » (processus permettant d'extraire la variabilité des données de manière structurée depuis le configurateur) et « web crawler » (processus explorant les configurations possibles et simulant les actions faisables par un utilisateur).

Dans un premier temps, l'utilisateur va définir un pattern vde❶ (variability data extraction) sous la forme HTML-like. Ce pattern est basé sur le code source HTML généré et exprimant la structure des données (ex. : le div ayant la classe « column2 » représente une balise de type image dont son url est l'attribut « src »). Une fois la structure extraite manuellement, elle est donnée au web wrapper qui pourra faire correspondre le code source du générateur avec le pattern vde reçu dans le but d'extraire les données proposées par ce dernier❷ (s'il existe des changements de pages, c'est le crawlPage❸ qui s'en chargera et demandera, au besoin, un nouveau vde). Le résultat est stocké, sous la forme de FM, dans un fichier xml. Au besoin, ces données peuvent être modifiées de manière automatique ou manuelle❹.

Ensuite, une suite de modèles TVL est générée❺ (chaque modèle représentant une étape de la configuration proposé par le web configurator sous la forme de FM). L'ensemble de ces fichiers TVL est alors importé dans l'outil FAMILIAR ❻ permettant ainsi une fusion de ces feature models en un FM final et complet.

L'article propose un schéma visuel de ce processus (Figure 43). Une implémentation et une série de discussions sont aussi proposées. Le processus est implémenté sous la forme d'application Java ou d'extension au module « Firebug » (module du navigateur Mozilla Firefox).

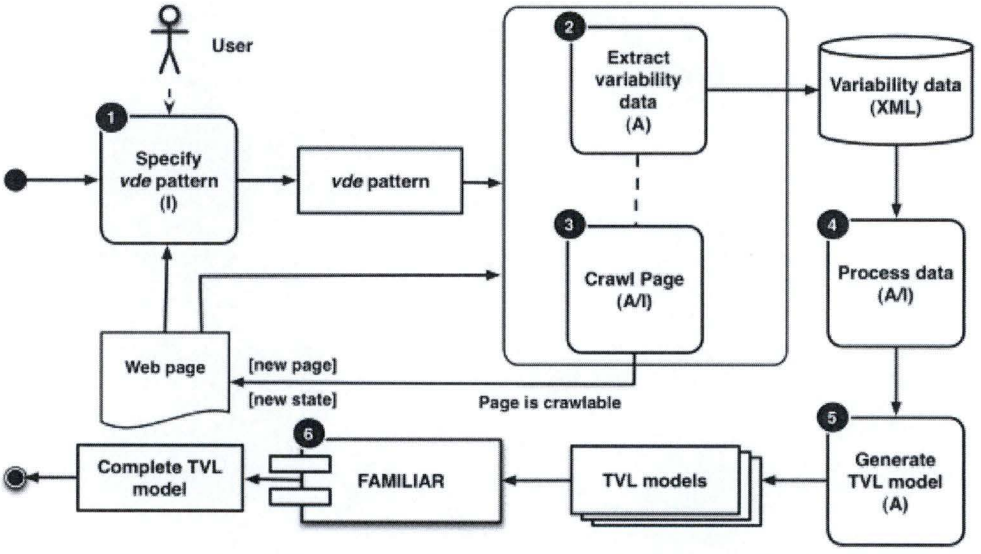


Figure 43 - Processus de rétro-ingénierie d'un "web configurator" (Ebrahim Khalil Abbasi 2014)

2 Analyse des publications

Une chose intéressante à faire ressortir dans ces 2 articles, est que tous deux se basent sur le code source HTML côté client. Il y a donc une grande dépendance vis-à-vis du navigateur qui, dans certains cas, est optimisé pour améliorer ce HTML (ex. : des balises non auto-fermées comme <br> peuvent être automatiquement remplacées par <br />). Dans le meilleur des cas, des changements mineurs et non influençant le processus peuvent apparaître. Il est à espérer que les navigateurs web futurs n'auront toujours pas d'impact sur ces techniques et qu'elles ne deviendront pas totalement dépendantes d'un navigateur web particulier (voire d'une version particulière).

Les 2 publications sont par contre très différentes dans le choix de la variabilité exprimée. Dans la première ([b]), la variabilité exprimée est celle du système et des features qu'il offre aux utilisateurs (des études de cas ont d'ailleurs été menées par les auteurs sur des sites Internet de grande taille proposant une importante variété de services tels que YouTube, Google et la BBC). Dans la seconde méthode, c'est l'expression de la variabilité d'objets physique par le biais d'un site système web qui est directement recherchée. Mais derrière cette approche, il est possible de voir, l'expression de la variabilité du configurateur web. En effet, la méthode est essentiellement basée sur ce que propose, comme options de l'objet, ce configurateur et donc implicitement, toutes les features possibles proposées par celui-ci. Par exemple, sur un véhicule de telle marque, tel modèle, une série d'options configurables sont proposées. Pour une autre marque et un autre modèle, d'autres options différentes peuvent être proposées. Pour ces 2 marques, le configurateur peut donc être construit comme une LPL dont chaque produit proposé serait composé de différentes features communes ou non. Un second exemple serait un configurateur web d'habits, il serait possible d'avoir une version du configurateur pour des chemises et un autre pour des pantalons. Des options de couleurs, matières, tailles ... sont proposées dans les 2 cas mais les features de choix du « type de col », « type de bouton de manchette » ... ne le seraient qu'uniquement dans le cas du configurateur de chemises.

Un 3e constat est que la littérature ne relate que très peu d'orientations pour ces domaines. Les directions à tester, comme par exemple, l'analyse depuis les traces d'un serveur web (typiquement « Apache http Server », « Microsoft IIS » ...) ou l'analyse de paquets TCP/IP sont encore des pistes à explorer dans les années à venir.

Conclusion

L'objectif principal de ce travail était principalement la découverte des techniques de rétro-ingénierie de systèmes web sous forme de lignes de produits logiciels. La recherche a été décomposée en 3 grandes parties.

Pour commencer, une description complète du contexte prometteur qui encadre et donne une émergence à ce sujet. Cette partie a permis la mise en évidence de l'importance de l'informatique et le point critique qu'elle représente pour la société actuelle. Une attention toute particulière a été retenue pour les systèmes de type web (utilisant de plus en plus le principe de réutilisabilité et dont le développement s'oriente vers des systèmes à haute variabilité comme les CMS configurables) dont les développements sont souvent réalisés par des start-ups très dynamiques. Ce dynamisme implique souvent un laisser-aller au niveau des analyses entraînant ainsi des surcoûts prévisibles lors des futures maintenances et évolutions. C'est dans ce cadre que les techniques de rétro-ingénierie proposées se font apprécier. Ces techniques étant encore peu connues, ce mémoire a dressé un état des lieux de ce domaine et été mené et présenté sous la forme d'une systematic mapping study. L'accent était particulièrement mis sur l'extraction de la variabilité étant donné son importance dans ce genre de systèmes.

L'état de l'art de ce mémoire a présenté, dans un premier chapitre, les lignes de produits logiciels et mis en évidence leurs origines récentes dans le milieu de l'informatique (bien que provenant de mécanismes industriels plus anciens). Il explique également les changements engendrés par rapport aux développements « traditionnels ». Le principal changement étant l'apparition des notions de variabilité et commonalities, permettant l'expression des différences et points communs (artéfacts réutilisables) d'un système. Cette variabilité a été analysée dans les détails. Les feature models (présentés la première fois par la méthode FODA), entièrement dédiés à l'expression de cette variabilité, ont également été représentés avec quelques-unes des techniques de validation existantes.

Un second chapitre a déterminé ce que peut apporter la rétro-ingénierie au sein de ce genre de systèmes particuliers comme : la génération de vues alternatives, la récupération d'informations perdues (ou non existantes ou incomplètes), etc. Il a été aussi possible d'y apprécier l'effet financier bénéfique que peuvent apporter ces méthodes. Une nouvelle piste a d'ailleurs été proposée quant à l'utilité de telles techniques sur la variabilité et l'avantage qu'elles pourraient apporter dans la détection de bugs par comparaison de modèles.

Une autre nouvelle piste ayant été proposée dans ce travail concerne l'extraction de la variabilité sur base du comportement d'un système implémenté : les feature transitions systems. Ces systèmes basés sur les transitions systems pourraient permettre l'expression exacte de cette variabilité. Malgré les recherches menées, aucune technique basée sur ce type de modèle n'a encore été relatée. L'idée est donc à mûrir et à développer dans les travaux futurs.

Au terme de cet état de l'art, une recherche de type systématique (« systematic mapping study ») a été menée. Cette méthode permet l'exploration d'un domaine ciblé afin d'en tirer

les voies déjà explorées ou au contraire, celles en manque de recherche. C'est cette recherche qui permet d'apporter cette pierre à l'édifice du contexte global de recherche de ce mémoire. Pour ce, des réponses aux questions suivantes ont été apportées : (1) Quelles sont les différentes techniques connues de RIV ? (2) Quelles sont les différentes sources d'entrée sur lesquelles se basent ces techniques ? (3) Quels sont les différents artéfacts de sortie produits par ces techniques ? (4) Quels sont les différents mécanismes connus permettant la gestion (l'expression et la manipulation) de la variabilité au sein des techniques de RIV ? (4 – sous-question) Quels sont les artéfacts particuliers intermédiaires (orientés variabilité) utilisés au sein de ces techniques ? (5) Existe-t-il des méthodes pour évaluer la couverture réelle d'une technique de rétro-ingénierie appliquée ? (6) Quels sont les types de publications, ainsi que leur évolution dans le temps, concernant la RIV de LPL ? (7) Existe-t-il des outils particuliers intervenant dans les mécanismes de RIV (exclusivement pour ou non) ?

Chaque étape de cette mapping study est détaillée au fur et à mesurée qu'effectuée permettant ainsi de montrer une marche à suivre complète et appliquée (fixée et validée par les co-promoteurs du mémoire). Pour ce faire, une série de « bonnes pratiques » provenant des publications de da Mota Silveira Neto et al., Devroey et al. ainsi que Petersen et al. ont été mises en œuvre.

A l'issue des multiples recherches réalisées sur les systèmes web, très peu de résultats ont été trouvés (2 publications). Le manque de publications sur ce sujet ne permettait pas d'avancer et de pouvoir tirer les bénéfices de cette technique. Il a donc été convenu d'élargir le champ de recherche à l'ensemble des systèmes informatiques.

Après cette phase de récupération et filtrage de publications, une phase importante de classification a été menée. L'ensemble de 72 publications récupérées (sur les 1807 publications brutes trouvées par les différentes méthodes de recherche) ont été analysées afin de classifier les mots clés représentant chaque publication au travers de multiples facettes. Les facettes ont émergé afin de répondre aux questions de recherches précédentes. Une fois ce classement effectué, des regroupements (et couplages) de données et d'interprétations des résultats ont ensuite été proposés.

Ces interprétations ont permis, dans un premier temps, de mettre clairement en évidence les différentes sources d'entrée et artéfacts de sortie des techniques découvertes. Il en ressort essentiellement 3 couples, étant, par ordre décroissant de fréquence d'utilisation : (a) code source vers modèles, (b) feature sets vers modèles et (c) diagrammes/modèles vers modèles, avec une claire et nette recherche de modélisation de feature models. Il est aussi ressorti que ces techniques puisent leurs sources d'entrée, soit depuis l'implémentation du système (dans 45.5 % des cas) soit depuis son analyse pour en reconstruire des modèles orientés variabilité (ex. : des listes ou matrices de configuration de produits). Quant aux voies permettant directement la modélisation comportementale prenant en compte la variabilité, elles sont presque inexistantes.
Il a été noté que ces techniques ne sont jamais nommées. Leurs utilisations s'effectuent donc en nommant les auteurs et publications les référençant.

Au-delà de ces utilisations, les analyses ont permis le listing d'une série de mécanismes exploités dans ces techniques (dont les plus connus sont les cross-tree constraints suivis des annotations de type ifdef-blocks du code source) ainsi que des artéfacts intermédiaires sur lesquels elles se basent (essentiellement : différentes formes de modèles, graphes et algorithmes heuristiques d'analyse).

A l'heure actuelle, aucune publication ne relate un moyen formel de vérifier la couverture réelle d'une technique par rapport au système qu'elle étudie. Les seules solutions proposées sont d'appliquer les techniques sur des systèmes connus, triviaux, déjà analysés ou s'il est possible, de vérifier leurs résultats auprès des développeurs. Une solution proposée dans ce mémoire est d'utiliser le mécanisme de correction des feature models pour en déduire un pourcentage d'erreurs.

Il a été objectivement possible de déterminer que les types de recherches (entre 2004 et 2015) les plus publiées sont des propositions de méthodes/approches suivies d'environ 40 % de publications d'expériences (cas d'utilisation). Un couplage a aussi permis de mettre en évidence, pour chaque couple de sources d'entrée et d'artéfacts de sortie, les types de publications dans lesquelles elles paraissent. Il en est ressorti un clair manque de publication d'évaluation et de validation des techniques. Cela confirme l'idée de début concernant la jeunesse et l'immaturité du sujet.

La suite et fin des analyses a permis de lister une série d'outils utilisés/présentés dans les publications. Ces outils sont, pour les exemples les plus connus, « FAMA tool suite », « FAMILAR » ou encore « Feature Commander ».

L'ensemble de informations pourront être, par exemple, directement utilisées par des praticiens (recherchant des méthodes précises) ou chercheurs (souhaitant, entre autres, découvrir les voies non encore explorées) dans les travaux futurs.

Afin de ne laisser aucune publication ignorée, 2 de celles-ci, entièrement orientées web (de 2014), récupérées dans le 1er objectif de la mapping study sont synthétisées et comparées. Une évidence est tout de suite apparue : l'une exprime la variabilité d'un système et la seconde, la variabilité d'un objet matériel à travers les systèmes web. Lors de l'analyse de cette seconde, un lien entre la méthode et la découverte de la variabilité dans le système en lui-même a été mis au jour. Cette différence et ce lien indiquent que de toutes récentes recherches pour les systèmes web sont en cours, et ce, selon différents axes.

Bibliographie

Références

Acher, Mathieu, et al. «On Extracting Feature Models From Product Descriptions.» *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '12*, 2012: 45-54.

Al-Msie'Deen, Ra'Fat. «Reverse Engineering Feature Models From Software Variants to Build Software Product Lines: REVPLINE Approach.» Université Montpellier, 25 Juin 2014.

Bachmann, Felix, et Len Bass. «Managing Variability in Software Architecture.» *Proceedings of the ACM SIGSOFT Symposium on Software Reusability (SSR'01)*, 2001: 126–132.

Batory, Don. «Feature Models, Grammars, and Propositional Formulas.» Dans *Software Product Lines*, 7-20. 2005.

Classen, Andreas, Cordy Maxime, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, et Jean-Franc¸ois Raskin. «Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking.» *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 2013: 1069-1089.

Cuevas, David Benavides. *On the automated analysis of software product lines using feature models.* Dissertation, Sevilla: The Department of Computer Languages and Systems ETSI Informática, 2007.

David Benavides, Sergio Segura and Antonio Ruiz-Cortés. «Automated Analysis of Feature Models 20 Years Later: A Literature Review*.» *Information Systems*, 2010: 615-636.

Devroey, Xavier, Gilles Perrouin, Axel Legay, Maxime Cordy, Pierre-Yves Schobbens, et Patrick Heymans. «Coverage Criteria for Behavioural Testing of Software Product Lines.» *Proceeding Part I of the Proceedings of the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - Volume 8802*, 2014: 336-350 .

Devroey, Xavier, Gilles Perrouin, et Pierre-Yves Schobbens. «Abstract test case generation for behavioural testing of software product lines.» *Proceeding SPLC '14 Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, 2014: 86-93.

Devroey, Xavier, Gilles Perrouin, Pierre-Yves Schobbens, Patrick Heymans, et Axel Legay. «State Machine Flattening: Mapping Study and Assessment.» *CoRR*, 2014.

Ebrahim Khalil Abbasi, Mathieu Acher, Patrick Heymans, Anthony Cleve. «Reverse Engineering Web Con.» *17th European Conference on Software Maintenance and Reengineering (CSMR)*, 2014.

Elkaim., W. El. *System family architecture glossary. Technical report.* ESAPS Project, 2001.

Elliot J. Chikofsky, James H. Cross II. «Reverse Engineering and Design Recorvery : A taxomany.» *IEEE*, 1990: 13-17.

fédération e-commerce et vente à distance. *CHIFFRES CLÉS | 2014* . Paris, 2014.

HACHICHI, GHANIA. *MODÈLE DE MESURE DE LA MAINTENANCE DE LOGICIEL.* 17 03 2011. (accès le 05 07, 2015).

Halmans, Günter, et Klaus Pohl. «Communicating the variability of a software-product family to customers.» *Proceedings of the Software and Systems Modeling* (Springer), 2003: 15-36.

Henard, Christopher, Mike Papadakis, Gilles Perrouin, Jacques Klein, et Yves Le Traon. «Towards Automated Testing and Fixing of Re-engineered Feature Models.» *Proceedings of the 2013 International Conference on Software Engineering*, 2013: 1245-1248.

HEYMANS, Patrick, et Jean-Christophe TRIGAUX. «Software Product Lines : State of the art .» Rapport de projet, Namur, 2003.

K. Czarnecki, S. Helsen, and U. Eisenecker. «Formalizing cardinality-based feature models and their specialization.» *Software Process: Improvement and Practice*, 2005: 7-29.

Kang, K., S. Cohen, J. Hess, W. Nowak, et S Peterson. «*Feature-Oriented Domain Analysis (FODA) Feasibility Study*» . Software Engineering Institute, Carnegie Mellon University, 1990.

Krueger, Charles W. «Requirements engineering for systems and software product lines.» *BigLever Software*, 2009.

L.Northrop, P. Clements and. «*Software Product Lines : Practices and Patterns*». Addison-Wesley, 2001.

Lai, M.D. Weiss and C.T. Robert. «*Software Product-Line Engineering : A FamilyBased*». Addison-Wisley, 1999.

M. Riebisch, K. B ¨ ollert, D. Streitferdt, and I. Philippow. «Extending feature diagrams with UML multiplicities.» *In 6th World Conference on Integrated Design & Process Technology (IDPT2002)*, 2002.

McGregor, John D., Linda M. Northrop, Salah Jarrad, et Klaus Pohl. «Initiating Software Product Lines.» *IEEE SOFTWARE*, 2002.

Mendeley. s.d. http://www.mendeley.com (accès le 02 06, 2015).

Parnas, D.L. «On the Design and Development of Program Families.» *Software Engineering, IEEE Transactions on*, 1976: 1-9.

Paulo Anselmo da Mota Silveira Neto, Ivan do Carmo Machado, John D. McGregor, Eduardo Santana de Almeida, et Silvio Romero de Lemos Meira. «A systematic mapping study of software product lines testing.» *Elsevier*, 2010.

Petersen, K., R. Feldt, S. Mujtaba, et M. Mattsson. «Systematic mapping studies in software engineering.» *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, s.d.

Priß, Dipl.-Math. Uta. «Relational Concept Analysis: Semantic Structures in Dictionaries and Lexical Databases1.» 1996.

Rekoff, M.G. «On reverse engineering.» *Systems, Man and Cybernetics, IEEE Transactions on*, 1985: 244 - 252.

Roel Wieringa, Neil Maiden, Nancy Mead, Colette Rolland. «Requirements engineering paper classification and evaluation criteria: a proposal and a discussion.» *Requirements Engineering* 11 (2006): 102-107.

Romdhane, Mohamed BEN. «Analyse des publications scientifiques : caractéristiques, structures et langages.» 1995-1996.

Rubin, Julia, et Marsha Chechik. «A Survey of Feature Location Techniques.» Dans *Domain Engineering*, de Iris Reinhartz-Berger, Arnon Sturm, Tony Clark, Sholom Cohen et Jorn Bettin, 29-58. Springer Berlin Heidelberg, 2013.

Schmid, Klaus, et Martin Verlage. «The Economic Impact of Product Line Adoption and Evolution.» *IEEE SOFTWARE*, 2002.

Schöpfel, Joachim. «Towards a Prague Definition of Grey Literature.» 2011.

Segura, Sergio, José A. Parejo, Robert M. Hierons, David Benavides, et Antonio Ruiz-Cortés. «Automated generation of computationally hard feature models using evolutionary algorithms.» *Expert Systems with Applications: An International Journal*, 2014: 3975-3992.

Sommerville, Ian. «*Software Engineering*». Ninth, 2009.

Technologies, DBI. *Extractor - Key Word Content Summary.* s.d. http://www.dbi-tech.com/ProductPage_Extractor.aspx (accès le 02 15, 2015).

Tilley, Thomas, Richard Cole1, Peter Becker1, et Peter Eklund2. «A Survey of Formal Concept Analysis Support for Software Engineering Activities.» s.d.

Translated Labs. *Terminology Extraction.* s.d. http://labs.translated.net/terminology-extraction/ (accès le 02 23, 2015).

Xavier Dolques, Marianne Huchard, Clementine Nebut, Hajer Saada. «Formal and Relational Concept Analysis approaches in Software Engineering: an overview and an application to learn model transformation patterns in examples.» *ICESE'11: First Virtual Workshop on Search-based Model-Driven Engineering*, 2011.

Zhang, Bo, et Martin Becker. «RECoVar: A Solution Framework towards Reverse Engineering Variability .» *4th International Workshop on Product LinE Approaches in Software Engineering*, 2013: 45-18.

Ziadi, Tewfik. «Les Lignes de Produits Logiciels(Software Product Lines).» UPMC, 2012.

ZIADI, Tewfik. «Manipulation de Lignes de Produits en UML.» IFSIC, Université de Rennes 1, 2004.

ziqizhan...@googlemail.com. *Java Automatic Term Extraction toolkit.* s.d. http://code.google.com/p/jatetoolkit/ (accès le 02 23, 2015).

Annexes

I. La forme de définition d'une feature dans une documentation de feature model

Name: <standard feature name>

Synonyms: <name> [FROM <source name>]

One or more synonyms may be defined, and the source of each name may optionally be included.
Description:
<textual description of the feature>

Consists Of <feature names> [{ optional | alternative }]

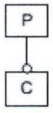
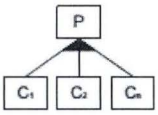
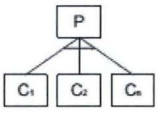
This information shows the hierarchical structure of features, and may be represented graphically.
Source:
<information source>

This information is used to produce a feature catalog. The source of information (e.g., standards, textbooks, existing systems) from which the feature is derived is included here.
Type: { compile-time | load-time | runtime }

[Mutually Exclusive With: <feature names>]
[Mandatory With: <feature names>]

Extrait venant du rapport technique de Kang (Kang, et al. 1990)

II. Contraintes dans les Feature models associées à la logique propositionnelle

Relationship		PL Mapping	Mobile Phone Example
MANDATORY		$P \leftrightarrow C$	MobilePhone $\leftrightarrow$ Calls MobilePhone $\leftrightarrow$ Screen
OPTIONAL		$C \rightarrow P$	GPS $\rightarrow$ MobilePhone Media $\rightarrow$ MobilePhone
OR		$P \leftrightarrow (C_1 \vee C_2 \vee ... \vee C_n)$	Media $\leftrightarrow$ (Camera $\vee$ MP3)
ALTERNATIVE		$(C_1 \leftrightarrow (\neg C_2 \wedge ... \wedge \neg C_n \wedge P)) \wedge$ $(C_2 \leftrightarrow (\neg C_1 \wedge ... \wedge \neg C_n \wedge P)) \wedge$ $(C_n \leftrightarrow (\neg C_1 \wedge \neg C_2 \wedge ... \wedge \neg C_{n-1} \wedge P))$	(Basic $\leftrightarrow$ (¬Color $\wedge$ ¬Highresolution $\wedge$ Screen)) $\wedge$ (Color $\leftrightarrow$ (¬Basic $\wedge$ ¬Highresolution $\wedge$ Screen)) $\wedge$ (Highresolution $\leftrightarrow$ (¬Basic $\wedge$ ¬Color $\wedge$ Screen))
IMPLIES		$A \rightarrow B$	Camera $\rightarrow$ Highresolution
EXCLUDES		$\neg(A \wedge B)$	¬(GPS $\wedge$ Basic)

III. Résultats des recherches dans les sources de données

Les recherches dans les tableaux verts sont des recherches sans les mots-clés « web » ou « internet ». Les tableaux en bleu prennent en compte ces mots-clés.

L'utilisation du mot-clé « extraction » sera retenue ou non (sélection basée sur une analyse de la différence en termes de nombre de résultats ainsi qu'une analyse sommaire des résultats afin d'estimer la pertinence des résultats trouvés avec ce mot-clé) en priorisant sa retenue.

❖ Google scholar

Chaîne tronquée automatiquement, une découpe est nécessaire.

allintitle:"reverse-engineering"|"reverse engineering"|"reversing"|"reverse behavior"|"reverse from" "product line"|"product family"|"feature-based"|"system family"|"software reusability"|"component-based"|"reuse context"	allintitle:"reverse-engineering"|"reverse engineering"|"reversing"|"reverse behavior"|"reverse from" "reuse oriented"|"feature-oriented"|"variability intensive system"|"configurable system"|"plugin-based"|"feature models"|"SPL"
43	16
allintitle:"get behavior"|"extraction"|"program comprehension"|"reverse architecting" "product line"|"product family"|"feature-based"|"system family"|"software reusability"|"component-based"|"reuse context"	allintitle:"get behavior"|"extraction"|"program comprehension"|"reverse architecting" "reuse oriented"|"feature-oriented"|"variability intensive system"|"configurable system"|"plugin-based"|"feature models"|"SPL"
541 (0 sans « extraction »)	30 (1 sans « extraction »)
89	

allintitle:"reverse-engineering"|"reverse engineering"|"reversing"|"reverse behavior"|"reverse from" "product line"|"product family"|"feature-based"|"system family"|"software reusability"|"component-based"|"reuse context" "web"|"internet"	allintitle:"reverse-engineering"|"reverse engineering"|"reversing"|"reverse behavior"|"reverse from" "reuse oriented"|"feature-oriented"|"variability intensive system"|"configurable system"|"plugin-based"|"feature models"|"SPL" "web"|"internet"
1	0
allintitle:"get behavior"|"extraction"|"program comprehension"|"reverse architecting" "product line"|"product family"|"feature-based"|"system family"|"software reusability"|"component-based"|"reuse context" "web"|"internet"	allintitle:"get behavior"|"extraction"|"program comprehension"|"reverse architecting" "reuse oriented"|"feature-oriented"|"variability intensive system"|"configurable system"|"plugin-based"|"feature models"|"SPL" "web"|"internet"
10 (0 sans le mot-clé « extraction »)	0
11	

❖ Microsoft Academic search

Une découpe rigoureuse suivant les règles des contraintes AND et OR est nécéssaire

title:("reverse-engineering" "product line")	title:("reverse-engineering" "product family")	title:("reverse-engineering" "feature-based")	title:("reverse-engineering" "system family")	title:("reverse-engineering" "software reusability")
title:("reverse engineering" "product line")	title:("reverse engineering" "product family")	title:("reverse engineering" "feature-based")	title:("reverse engineering" "system family")	title:("reverse engineering" "software reusability")
title:("reversing" "product line")	title:("reversing" "product family")	title:("reversing" "feature-based")	title:("reversing" "system family")	title:("reversing" "software reusability")
title:("reverse behavior" "product line")	title:("reverse behavior" "product family")	title:("reverse behavior" "feature-based")	title:("reverse behavior" "system family")	title:("reverse behavior" "software reusability")

title:("reverse from" "product line")	title:("reverse from" "product family")	title:("reverse from" "feature-based")	title:("reverse from" "system family")	title:("reverse from" "software reusability")
title:("get behavior" "product line")	title:("get behavior" "product family")	title:("get behavior" "feature-based")	title:("get behavior" "system family")	title:("get behavior" "software reusability")
title:("extraction" "product line")	title:("extraction" "product family")	title:("extraction" "feature-based")	title:("extraction" "system family")	title:("extraction" "software reusability")
title:("program comprehension" "product line")	title:("program comprehension" "product family")	title:("program comprehension" "feature-based")	title:("program comprehension" "system family")	title:("program comprehension" "software reusability")
title:("reverse architecting" "product line")	title:("reverse architecting" "product family")	title:("reverse architecting" "feature-based")	title:("reverse architecting" "system family")	title:("reverse architecting" "software reusability")
title:("reverse-engineering" "component-based")	title:("reverse-engineering" "reuse context")	title:("reverse-engineering" "reuse oriented")	title:("reverse-engineering" "feature-oriented")	title:("reverse-engineering" "variability intensive system")
title:("reverse engineering" "component-based")	title:("reverse engineering" "reuse context")	title:("reverse engineering" "reuse oriented")	title:("reverse engineering" "feature-oriented")	title:("reverse engineering" "variability intensive system")
title:("reversing" "component-based")	title:("reversing" "reuse context")	title:("reversing" "reuse oriented")	title:("reversing" "feature-oriented")	title:("reversing" "variability intensive system")
title:("reverse behavior" "component-based")	title:("reverse behavior" "reuse context")	title:("reverse behavior" "reuse oriented")	title:("reverse behavior" "feature-oriented")	title:("reverse behavior" "variability intensive system")
title:("reverse from" "component-based")	title:("reverse from" "reuse context")	title:("reverse from" "reuse oriented")	title:("reverse from" "feature-oriented")	title:("reverse from" "variability intensive system")
title:("get behavior" "component-based")	title:("get behavior" "reuse context")	title:("get behavior" "reuse oriented")	title:("get behavior" "feature-oriented")	title:("get behavior" "variability intensive system")
title:("extraction" "component-based")	title:("extraction" "reuse context")	title:("extraction" "reuse oriented")	title:("extraction" "feature-oriented")	title:("extraction" "variability intensive system")
title:("program comprehension" "component-based")	title:("program comprehension" "reuse context")	title:("program comprehension" "reuse oriented")	title:("program comprehension" "feature-oriented")	title:("program comprehension" "variability intensive system")
title:("reverse architecting" "component-based")	title:("reverse architecting" "reuse context")	title:("reverse architecting" "reuse oriented")	title:("reverse architecting" "feature-oriented")	title:("reverse architecting" "variability intensive system")
title:("reverse-engineering" "configurable system")	title:("reverse-engineering" "plugin-based")	title:("reverse-engineering" "feature models")	title:("reverse-engineering" "SPL")	
title:("reverse engineering" "configurable system")	title:("reverse engineering" "plugin-based")	title:("reverse engineering" "feature models")	title:("reverse engineering" "SPL")	
title:("reversing" "configurable system")	title:("reversing" "plugin-based")	title:("reversing" "feature models")	title:("reversing" "SPL")	
title:("reverse behavior" "configurable system")	title:("reverse behavior" "plugin-based")	title:("reverse behavior" "feature models")	title:("reverse behavior" "SPL")	
title:("reverse from" "configurable system")	title:("reverse from" "plugin-based")	title:("reverse from" "feature models")	title:("reverse from" "SPL")	
title:("get behavior" "configurable system")	title:("get behavior" "plugin-based")	title:("get behavior" "feature models")	title:("get behavior" "SPL")	
title:("extraction" "configurable system")	title:("extraction" "plugin-based")	title:("extraction" "feature models")	title:("extraction" "SPL")	

title:("program comprehension" "configurable system") title:("reverse architecting" "configurable system")	title:("program comprehension" "plugin-based") title:("reverse architecting" "plugin-based")	title:("program comprehension" "feature models") title:("reverse architecting" "feature models")	title:("program comprehension" "SPL") title:("reverse architecting" "SPL")
126			

❖ Dblp – computer science bibliography

reverse|behavior$| product-line|family|reusability|reusable|product|reuse|SPL|configurable|variability|feature|plugin
14

reverse|behavior$| product-line|family|reusability|reusable|product|reuse|SPL|configurable|variability|feature web|internet
1

❖ ScienceDirect

Title-Abstr-Key(("reverse engineering" OR "reverse-engineering" OR "reverse architecting" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension") AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL"))
195 (21 sans « extraction »)

Title-Abstr-Key(("reverse engineering" OR "reverse-engineering" OR "reverse architecting" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension") AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet"))
9 (0 sans « extraction »)

❖ ACM – Digital Library

(Abstract:"reverse engineering" or Abstract:"reverse-engineering" or Abstract:"reverse architecting" or Abstract:"reverse behavior" or Abstract:"reversing" or Abstract:"reverse from" or Abstract:"get behavior" or Abstract:"extraction" or Abstract:"program comprehension") and (Abstract:"product line" or Abstract:"product family" or Abstract:"system family" or Abstract:"feature-based" or Abstract:"software reusability" or Abstract:"product-line" or Abstract:"component-based" or Abstract:"reuse context" or Abstract:"reuse oriented" or Abstract:"feature-oriented" or Abstract:"variability intensive system" or Abstract:"configurable system" or Abstract:"plugin-based" or Abstract:"feature models" or Abstract:"SPL")		
Abstract : 382 (59 sans « extraction »)	Keywords : 55 (43 sans « extraction »)	Title : 31 (10 sans « extraction »)
112		

(Abstract:"reverse engineering" or Abstract:"reverse-engineering" or Abstract:"reverse architecting" or Abstract:"reverse behavior" or Abstract:"reversing" or Abstract:"reverse from" or Abstract:"get behavior" or Abstract:"extraction" or Abstract:"program comprehension") and (Abstract:"product line" or Abstract:"product family" or Abstract:"system family" or Abstract:"feature-based" or Abstract:"software reusability" or Abstract:"product-line" or Abstract:"component-based" or Abstract:"reuse context" or

Abstract:"reuse oriented" or Abstract:"feature-oriented" or Abstract:"variability intensive system" or Abstract:"configurable system" or Abstract:"plugin-based" or Abstract:"feature models" or Abstract:"SPL") and (Abstract:"web" OR Abstract:"internet")		
Abstract : 5	Keywords : 3	Title : 1
9		

❖ Springer Link

(dc.title ="reverse engineering" or dc.title ="behavior") and (dc.title ="product-line" or dc.title ="family" or dc.title ="reusability" or dc.title ="reusable" or dc.title ="product" or dc.title ="reuse" or dc.title ="SPL" or dc.title ="configurable" or dc.title ="variability" or dc.title ="feature" or dc.title ="plugin")
9

❖ The Collection of Computer Science Bibliographies

+(ti:"reverse engineering" OR ti:"reverse-engineering" OR ti:"reverse behavior" OR ti:"reversing" OR ti:"reverse from" OR ti:"get behavior" OR ti:"extraction" OR ti:"program comprehension" OR ti:"reverse architecting")+(ti:"product line" OR ti:"product family" OR ti:"system family" OR ti:"feature-based" OR ti:"software reusability" OR ti:"product-line" OR ti:"component-based" OR ti:"reuse context" OR ti:"reuse oriented" OR ti:"feature-oriented" OR ti:"variability intensive system" OR ti:"configurable system" OR ti:"plugin-based" OR "feature models" OR ti:"SPL")
79 (36 sans « extraction »)

+(ti:"reverse engineering" OR ti:"reverse-engineering" OR ti:"reverse behavior" OR ti:"reversing" OR ti:"reverse from" OR ti:"get behavior" OR ti:"extraction" OR ti:"program comprehension" OR ti:"reverse architecting")+(ti:"product line" OR ti:"product family" OR ti:"system family" OR ti:"feature-based" OR ti:"software reusability" OR ti:"product-line" OR ti:"component-based" OR ti:"reuse context" OR ti:"reuse oriented" OR ti:"feature-oriented" OR ti:"variability intensive system" OR ti:"configurable system" OR ti:"plugin-based" OR "feature models" OR ti:"SPL")+("web" OR "internet")
5 (2 sans « extraction »)

❖ System for Information on Grey Literature in Europe

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
12

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet")
1

❖ Directory of open access journals

Chaîne trop longue, une découpe est nécessaire.

("reverse-engineering" OR "reverse engineering") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL")		
Abstract : 19	Keywords : 4	Title : 3
("reversing" OR "reverse behavior") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL")		
Abstract : 0	Keywords : 0	Title : 0
("reverse from" OR "get behavior") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL")		
Abstract : 1	Keywords : 0	Title : 0
("extraction" OR "program comprehension") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL")		
Abstract : 3	Keywords : 1	Title : 0
("reverse architecting") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL")		
Abstract : 0	Keywords : 0	Title : 0
31		

("reverse-engineering" OR "reverse engineering") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL") AND ("web" OR "internet")		
Abstract : 3	Keywords : 0	Title : 0
("reversing" OR "reverse behavior") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL") AND ("web" OR "internet")		
Abstract : 0	Keywords : 0	Title : 0
("reverse from" OR "get behavior") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL") AND ("web" OR "internet")		
Abstract : 0	Keywords : 0	Title : 0
("extraction" OR "program comprehension") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL") AND ("web" OR "internet")		
Abstract : 0	Keywords : 0	Title : 0
("reverse architecting") AND ("product" OR "system family" OR "feature" OR "reuse" OR "component-based" OR "variability" OR "configurable system" OR "plugin-based" OR "SPL") AND ("web" OR "internet")		
Abstract : 0	Keywords : 0	Title : 0
3		

❖ The Directory of Open Access Repositories

allintitle:(("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL"))
214 (15 sans « extraction ») (439000 sans allintitle)

allintitle:(("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" or "internet"))
215 (15 sans « extraction ») (423000 sans allintitle)

❖ Centre pour la Communication Scientifique Directe

Chaîne trop longue, une découpe est nécessaire.

("reverse-engineering" OR "reverse engineering" OR "reversing" OR "reverse behavior" OR "reverse from") AND("product line" OR "product family" OR "feature-based" OR "system family" OR "software reusability" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented")	("reverse-engineering" OR "reverse engineering" OR "reversing" OR "reverse behavior" OR "reverse from") AND("variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
14	12
("get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting") AND ("product line" OR "product family" OR "feature-based" OR "system family" OR "software reusability" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented")	("get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting") AND ("variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
22 (0 sans « extraction »)	9
48	

("reverse-engineering"OR"reverse engineering"OR"reversing"OR"reverse behavior"OR"reverse from") AND("product line"OR"product family"OR"feature-based"OR"system family"OR"software reusability"OR"component-based"OR"reuse context"OR"reuse oriented"OR"feature-oriented")AND("web"OR"internet")	("reverse-engineering"OR"reverse engineering"OR"reversing"OR"reverse behavior"OR"reverse from") AND("variability intensive system"OR"configurable system"OR"plugin-based"OR"feature models"OR"SPL")AND("web"OR"internet")
0	1
("get behavior"OR"extraction"OR"program comprehension"OR"reverse architecting")AND("product line"OR"product family"OR"feature-based"OR"system family"OR"software reusability"OR"component-based"OR"reuse context"OR"reuse oriented"OR"feature-oriented")AND("web"OR"internet")	("get behavior"OR"extraction"OR"program comprehension"OR"reverse architecting") AND ("variability intensive system"OR"configurable system"OR"plugin-based"OR"feature models"OR"SPL")AND("web"OR"internet")
1	2
4	

❖ RefSeer: A Citation Recommendation System

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
150

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet")
150

❖ MIT Open Access Articles

((abstract:("reverse-engineering" OR "reverse engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")) AND (abstract:("product line" OR "product family" OR "system family" OR "software reusability" OR "product-line" OR "architecture" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-Oriented" OR "Variability Intensive System" OR "configurable" OR "plugin-based" OR "SPL")))	((title:("reverse-engineering" OR "reverse engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")) AND (title:("product line" OR "product family" OR "system family" OR "software reusability" OR "product-line" OR "architecture" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-Oriented" OR "Variability Intensive System" OR "configurable" OR "plugin-based" OR "SPL")))
206 (11 avec « AND (abstract:software) »)	4
15	

((abstract:("reverse-engineering" OR "reverse engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")) AND (abstract:("product line" OR "product family" OR "system family" OR "software reusability" OR "product-line" OR "architecture" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-Oriented" OR "Variability Intensive System" OR "configurable" OR "plugin-based" OR "SPL")) AND (abstract :("web" OR "internet")))	((title:("reverse-engineering" OR "reverse engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")) AND (title:("product line" OR "product family" OR "system family" OR "software reusability" OR "product-line" OR "architecture" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-Oriented" OR "Variability Intensive System" OR "configurable" OR "plugin-based" OR "SPL") AND (title :("web" OR "internet")))
3	0
3	

❖ ISU Computer Science Archives

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
194 (65 sans « extraction »)

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet")
119 (34 sans « extraction »)

❖ Cornell Universiry Library

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
97 sans « extraction » (des milliers avec « extraction »)

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet")
47 sans « extraction » (240 avec « extraction »)

❖ CiteSeerX

title:(("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL"))	abstract:((("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")))
22	971 (83 sans « extraction » et 34 avec « software »)
56	

title:(("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet"))	abstract:((("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet")))
0	40 (2 sans « extraction »)
40	

❖ **Electronic Theses and Dissertations website of the Katholieke Hogeschool Kempen**

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
3

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet")
3

❖ **Repositorio institucional de la UNLP**

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
4293 (96 avec « AND ("software engineering") »)

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet")
96 (13 avec « AND ("software engineering") »)

❖ **FreeSearch Toward computer science ideas**

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
125 (18 avec « software », « software engineering » ne retourne rien)

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet")
2 (0 avec « software », « software engineering » ne retourne rien)

❖ L'IEEE Computer Society

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
71 via l'option Google (0 via la recherche interne)

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet")
64 via l'option Google (0 via la recherche interne)

❖ IEEE Xplore Digital Library

Chaîne trop longue, une découpe est nécessaire.

("Document Title":"reverse-engineering" OR "Document Title":"reverse engineering" OR "Document Title":"reversing" OR "Document Title":"reverse behavior" OR "Document Title":"reverse from") AND("Document Title":"product line" OR "Document Title":"product family" OR "Document Title":"feature-based" OR "Document Title":"system family" OR "Document Title":"software reusability" OR "Document Title":"component-based" OR "Document Title":"reuse context" OR "Document Title":"reuse oriented" OR "Document Title":"feature-oriented")	("Abstract":"reverse-engineering" OR "Abstract":"reverse engineering" OR "Abstract":"reversing" OR "Abstract":"reverse behavior" OR "Abstract":"reverse from") AND("Abstract":"product line" OR "Abstract":"product family" OR "Abstract":"feature-based" OR "Abstract":"system family" OR "Abstract":"software reusability" OR "Abstract":"component-based" OR "Abstract":"reuse context" OR "Abstract":"reuse oriented" OR "Abstract":"feature-oriented")
7	38
("Document Title":"reverse-engineering" OR "Document Title":"reverse engineering" OR "Document Title":"reversing" OR "Document Title":"reverse behavior" OR "Document Title":"reverse from") AND("Document Title":"variability intensive system" OR "Document Title":"configurable system" OR "Document Title":"plugin-based" OR "Document Title":"feature models" OR "Document Title":"SPL")	("Abstract":"reverse-engineering" OR "Abstract":"reverse engineering" OR "Abstract":"reversing" OR "Abstract":"reverse behavior" OR "Abstract":"reverse from") AND("Abstract":"variability intensive system" OR "Abstract":"configurable system" OR "Abstract":"plugin-based" OR "Abstract":"feature models" OR "Abstract":"SPL")
2	6
("Document Title":"get behavior" OR "Document Title":"extraction" OR "Document Title":"program comprehension" OR "Document Title":"reverse architecting") AND ("Document Title":"product line" OR "Document Title":"product family" OR "Document Title":"feature-based" OR "Document Title":"system family" OR "Document Title":"software reusability" OR "Document Title":"component-based" OR "Document Title":"reuse context" OR "Document Title":"reuse oriented" OR "Document Title":"feature-oriented")	("Abstract":"get behavior" OR "Abstract":"extraction" OR "Abstract":"program comprehension" OR "Abstract":"reverse architecting") AND ("Abstract":"product line" OR "Abstract":"product family" OR "Abstract":"feature-based" OR "Abstract":"system family" OR "Abstract":"software reusability" OR "Abstract":"component-based" OR "Abstract":"reuse context" OR "Abstract":"reuse oriented" OR "Abstract":"feature-oriented")
29 (0 sans « extraction »)	342 (7 sans « extraction »)
("Document Title":"get behavior" OR "Document Title":"extraction" OR "Document Title":"program comprehension" OR "Document Title":"reverse architecting") AND ("Document Title":"variability intensive system" OR "Document Title":"configurable system" OR "Document	("Abstract":"get behavior" OR "Abstract":"extraction" OR "Abstract":"program comprehension" OR "Abstract":"reverse architecting") AND ("Abstract":"variability intensive system" OR "Abstract":"configurable system" OR "Abstract":"plugin-based" OR "Abstract":"feature models" OR "Abstract":"SPL")

Title":"plugin-based" OR "Document Title":"feature models" OR "Document Title":"SPL")	
0	21
111	

("Document Title":"reverse-engineering" OR "Document Title":"reverse engineering" OR "Document Title":"reversing" OR "Document Title":"reverse behavior" OR "Document Title":"reverse from") AND("Document Title":"product line" OR "Document Title":"product family" OR "Document Title":"feature-based" OR "Document Title":"system family" OR "Document Title":"software reusability" OR "Document Title":"component-based" OR "Document Title":"reuse context" OR "Document Title":"reuse oriented" OR "Document Title":"feature-oriented") AND ("web" OR "internet")	("Abstract":"reverse-engineering" OR "Abstract":"reverse engineering" OR "Abstract":"reversing" OR "Abstract":"reverse behavior" OR "Abstract":"reverse from") AND("Abstract":"product line" OR "Abstract":"product family" OR "Abstract":"feature-based" OR "Abstract":"system family" OR "Abstract":"software reusability" OR "Abstract":"component-based" OR "Abstract":"reuse context" OR "Abstract":"reuse oriented" OR "Abstract":"feature-oriented") AND ("web" OR "internet")
0	8
("Document Title":"reverse-engineering" OR "Document Title":"reverse engineering" OR "Document Title":"reversing" OR "Document Title":"reverse behavior" OR "Document Title":"reverse from") AND("Document Title":"variability intensive system" OR "Document Title":"configurable system" OR "Document Title":"plugin-based" OR "Document Title":"feature models" OR "Document Title":"SPL") AND ("web" OR "internet")	("Abstract":"reverse-engineering" OR "Abstract":"reverse engineering" OR "Abstract":"reversing" OR "Abstract":"reverse behavior" OR "Abstract":"reverse from") AND("Abstract":"variability intensive system" OR "Abstract":"configurable system" OR "Abstract":"plugin-based" OR "Abstract":"feature models" OR "Abstract":"SPL") AND ("web" OR "internet")
0	0
("Document Title":"get behavior" OR "Document Title":"extraction" OR "Document Title":"program comprehension" OR "Document Title":"reverse architecting") AND ("Document Title":"product line" OR "Document Title":"product family" OR "Document Title":"feature-based" OR "Document Title":"system family" OR "Document Title":"software reusability" OR "Document Title":"component-based" OR "Document Title":"reuse context" OR "Document Title":"reuse oriented" OR "Document Title":"feature-oriented") AND ("web" OR "internet")	("Abstract":"get behavior" OR "Abstract":"extraction" OR "Abstract":"program comprehension" OR "Abstract":"reverse architecting") AND ("Abstract":"product line" OR "Abstract":"product family" OR "Abstract":"feature-based" OR "Abstract":"system family" OR "Abstract":"software reusability" OR "Abstract":"component-based" OR "Abstract":"reuse context" OR "Abstract":"reuse oriented" OR "Abstract":"feature-oriented") AND ("web" OR "internet")
6	17 (0 sans « extraction »)
("Document Title":"get behavior" OR "Document Title":"extraction" OR "Document Title":"program comprehension" OR "Document Title":"reverse architecting") AND ("Document Title":"variability intensive system" OR "Document Title":"configurable system" OR "Document Title":"plugin-based" OR "Document Title":"feature models" OR "Document Title":"SPL") AND ("web" OR "internet")	("Abstract":"get behavior" OR "Abstract":"extraction" OR "Abstract":"program comprehension" OR "Abstract":"reverse architecting") AND ("Abstract":"variability intensive system" OR "Abstract":"configurable system" OR "Abstract":"plugin-based" OR "Abstract":"feature models" OR "Abstract":"SPL") AND ("web" OR "internet")
0	0
31	

❖ The Institution of Engineering and Technology

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL")
2 via Title pour contrainte technique

("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet")
0 via Title pour contrainte technique

❖ Mendeley

abstract:(("reverse engineering" OR "reverse-engineering" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension" OR "reverse architecting")AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL"))		
Abstract : 736 (13 avec AND ("software engineering")	Keywords : 7	Title : 58
65		

abstract:(("reverse engineering" OR "reverse-engineering" OR "reverse architecting" OR "reverse behavior" OR "reversing" OR "reverse from" OR "get behavior" OR "extraction" OR "program comprehension") AND ("product line" OR "product family" OR "system family" OR "feature-based" OR "software reusability" OR "product-line" OR "component-based" OR "reuse context" OR "reuse oriented" OR "feature-oriented" OR "variability intensive system" OR "configurable system" OR "plugin-based" OR "feature models" OR "SPL") AND ("web" OR "internet"))		
Abstract : 29 (5 sans « extraction »)	Keywords : 0	Title : 2 (0 sans « extraction »)
31		

IV. Références des résultats de la recherche avec "web" et "internet"

[a] Y. Ma and B. Jin, "Application of symbol feature-based HMM in web information extraction," in *2009 WRI World Congress on Computer Science and Information Engineering, CSIE 2009*, 2009, vol. 4, pp. 173–177.

[b] S. Marciuska, C. Gencel, and P. Abrahamsson, "Automated feature identification in web applications," *Lecture Notes in Business Information Processing*, vol. 166 LNBIP, pp. 100–114, Nov. 2014.

[c] H. M. Kienle and H. a. Müller, "A WSAD-Based fact extractor for J2EE web projects," in *Proceedings - 9th IEEE International Symposium on Web Site Evolution, WSE 2007*, 2007, pp. 57–64.

[d] W. Liu, P. Ren, K. Liu, and H. Duan, "Behavior-Based Malware Analysis and Detection," in *2011 First International Workshop on Complexity and Data Mining*, 2011, pp. 39–42.

[e] S. Sen, *Découverte automatique de modèles effectifs*. Université Rennes 1, 2010.

[f] C. Kastner, S. Apel, and M. Kuhlemann, "Granularity in software product lines," *2008 ACM/IEEE 30th International Conference on Software Engineering*, pp. 311–320, 2008.

[g] K. F. Harry C. Li, Shriram Krishnamurthi, "Interfaces for Modular Feature Verification."

[h] R. Sandhu, "Reverse {Engineering} in {Web} {Designing}:- {An} {Emerging} {Trend}," *International Journal of Advanced Research in Computer Science*.

[i] E. K. Abbasi, M. Acher, P. Heymans, and A. Cleve, "*Reverse engineering web configurators*.", 17th European Conference on Software Maintenance and Reengineering (CSMR), IEEE, 2014.

[j] T. Th, D. Batory, and C. K, "To appear ICSE 2009 Reasoning about Edits to Feature Models," 2009.

V. Références des résultats de la recherche (hors recherche approfondie)

[a] R. Al-Msie'Deen, A. D. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, "A methodology to recover feature models from object-oriented source code," *VARY'2012: VARiability for You*. 30-Oct-2012.

[b] M. T. Valente, V. Borges, and L. Passos, "A semi-automatic approach for extracting software product lines," *IEEE Trans. Softw. Eng.*, vol. 38, no. 4, pp. 737–754, Jul. 2012.

[c] R. Al-msie, a D. Seriai, M. Huchard, and C. Urtado, "An approach to recover feature models from object-oriented source code," *Journée Lignes de Produits*. pp. 1–12, 06-Nov-2012.

[d] R. E. Lopez-Herrejon, L. Linsbauer, J. a. Galindo, J. a. Parejo, D. Benavides, S. Segura, and A. Egyed, "An assessment of search-based techniques for reverse engineering feature models," *J. Syst. Softw.*, vol. 103, pp. 353–369, May 2014.

[e] C. Luna and A. Gonzalez, "Behavior specification of product lines via feature models and UML statecharts with variabilities," in *Proceedings - International Conference of the Chilean Computer Science Society, SCCC*, 2008, pp. 32–41.

[f] T. Ziadi, T. Ziadi, L. Hélouët, L. Hélouët, J.-M. Jézéquel, and J.-M. Jézéquel, "Behaviors Generation From Product Lines Requirements," *Banking*, 2004.

[g] G. Bécan, M. Acher, B. Baudry, and S. Ben Nasr, "Breathing ontological knowledge into feature model synthesis: an empirical study," *Empir. Softw. Eng.*, p. 51, Mar. 2015.

[h] B. Zhang and M. Becker, "Code-based variability model extraction for software product line improvement," in *ACM International Conference Proceeding Series*, 2012, vol. 2, pp. 91–98.

[i] A. E. El Hamdouni, a. D. Seriai, and M. Huchard, "Component-based architecture recovery from object oriented systems via relational concept analysis," *CEUR Workshop Proceedings*, 2010. [Online]. Available: http://ceur-ws.org/Vol-672/paper23.pdf. [Accessed: 23-Mar-2015].

[j] A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, and G. Saval, "Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis," in *15th IEEE International Requirements Engineering Conference (RE 2007)*, 2007, pp. 243–253.

[k] J. Feigenspan, C. Kästner, S. Apel, J. Liebig, M. Schulze, R. Dachselt, M. Papendieck, T. Leich, and G. Saake, "Do background colors improve program comprehension in the #ifdef hell?," *Empir. Softw. Eng.*, vol. 18, no. 4, pp. 699–745, May 2013.

[l] S. She, U. Ryssel, N. Andersen, A. Wąsowski, and K. Czarnecki, "Efficient synthesis of feature models," in *Information and Software Technology*, 2014, vol. 56, no. 9, pp. 1122–1143.

[m] O. Greevy, "Enriching Reverse Engineering with Feature Analysis," 2007. [Online]. Available: http://scg.unibe.ch/archive/phd/greevy-phd.pdf. [Accessed: 23-Mar-2015].

[n] M. V. Couto, M. T. Valente, and E. Figueiredo, "Extracting Software Product Lines: A Case Study Using Conditional Compilation," *2011 15th European Conference on Software Maintenance and Reengineering*, 2011. [Online]. Available: http://academic.research.microsoft.com/Publication/39260876/extracting-software-product-lines-a-case-study-using-conditional-compilation. [Accessed: 23-Mar-2015].

[o] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire, "Extraction and evolution of architectural variability models in plugin-based systems," *Softw. Syst. Model.*, vol. 13, no. 4, pp. 1–28, Jul. 2013.

[p] B. Zhang, "Extraction and improvement of conditionally compiled product line code," in *IEEE International Conference on Program Comprehension*, 2012, pp. 257–258.

[q] U. Ryssel, J. Ploennigs, and K. Kabitzsch, "Extraction of feature models from formal contexts," in *Proceedings of the 15th International Software Product Line Conference on - SPLC '11*, 2011, p. 1.

[r] S. Apel and D. Beyer, "Feature cohesion in software product lines: an exploratory study," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 421–430.

[s] K. Czarnecki and A. Wasowski, "Feature diagrams and logics: There and back again," *Proc. - 11th Int. Softw. Prod. Line Conf. SPLC 2007*, pp. 23–32, Sep. 2007.

[t] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: A taxonomy and survey," *Journal of software: Evolution and Process*, 2013. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.380.6285&rep=rep1&type=pdf. [Accessed: 04-Apr-2015].

[u] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans, "Feature Model Extraction from Large Collections of Informal Product Descriptions," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 2013, pp. 290–300.

[v] R. Stoiber and M. Glinz, "Feature unweaving: Efficient variability extraction and specification for emerging software product lines," in *2010 4th International Workshop on Software Product Management, IWSPM 2010*, 2010, pp. 53–62.

[w] V. Alves, F. Calheiros, V. Nepomuceno, A. Menezes, S. Soares, and P. Borba, "FLiP: Managing software product line extraction and reaction with aspects," in *Proceedings - 12th International Software Product Line Conference, SPLC 2008*, 2008, p. 354.

[x] J. Rubin and M. Chechik, "Locating distinguishing features using diff sets," *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on*, 2012. [Online]. Available: http://www.cs.toronto.edu/~chechik/pubs/ase12.pdf. [Accessed: 04-Apr-2015].

[y] J. Kofroň, F. Plášil, and O. Šerý, "Modes in component behavior specification via EBP and their application in product lines," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 31–41, Jan. 2009.

[z] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions," in *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems - VaMoS '12*, 2012, pp. 45–54.

[aa] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, *On extracting feature models from sets of valid feature combinations*, vol. 7793. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[ab] J. Feigenspan, "Program Comprehension of Feature-Oriented Software Development." [Online]. Available: http://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/auto/Feigenspan11.pdf. [Accessed: 20-Mar-2015].

[ac] L. P. T. C. M. F. Rubira, "Public Health Complaint Software Product Line," 2011.

[ad] B. Zhang and M. Becker, "RECoVar: A solution framework towards reverse engineering variability," in *2013 4th International Workshop on Product LinE Approaches in Software Engineering, PLEASE 2013 - Proceedings*, 2013, pp. 45–48.

[ae] H. Eyal-Salman, A.-D. Seriai, C. Dony, and R. Al-msie'deen, "Recovering traceability links between feature models and source code of product variants," in *Proceedings of the VARiability for You Workshop on Variability Modeling Made Useful for Everyone - VARY '12*, 2012, pp. 21–25.

[af] S. Duszynski and M. Becker, "Recovering variability information from the source code of similar software products," *2012 3rd Int. Work. Prod. LinE Approaches Softw. Eng. PLEASE 2012 - Proc.*, pp. 37–40, Jun. 2012.

[ag] A. Olszak and B. Nørregaard Jørgensen, "Remodularizing Java programs for improved locality of feature implementations in source code," *Sci. Comput. Program.*, vol. 77, no. 3, pp. 131–151, Mar. 2012.

[ah] D. Kim, W. N. Sumner, X. Zhang, D. Xu, and H. Agrawal, "Reuse-oriented reverse engineering of functional components from x86 binaries," in *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, 2014, pp. 1128–1139.

[ai] L. D. Mathieu Acher, Anthony Cleve, "Reverse Engineering Architectural Feature Models," *Proc. 5th Eur. Conf. Softw. Archit. (ECSA'11}*, 2011.

[aj] B. Zhang and M. Becker, "Reverse Engineering Complex Feature Correlations for Product Line Configuration Improvement," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2014, pp. 320–327.

[ak] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Reverse engineering feature models," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 461–470.

[al] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "Reverse engineering feature models from programs' feature sets," in *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2011, pp. 308–312.

[am] R. A.-M. 'Deen, M. Huchard, A.-D. Seriai, C. Urtado, and S. Vauttier, "Reverse Engineering Feature Models from Software Configurations using Formal Concept Analysis," *CLA 2014: Eleventh International Conference on Concept Lattices and Their Applications*, vol. 1252. pp. 95–106, 01-Oct-2014.

[an] G. Bécan, "Reverse Engineering Feature Models in the Real." INRIA-IRISA Rennes Bretagne Atlantique, équipe TRISKELL, p. 40, 25-Jun-2013.

[ao] A. E. Roberto E. Lopez-Herrejon1, José A. Galindo2, David Benavides2, Sergio Segura2, *Reverse engineering feature models with evolutionary algorithms: an exploratory study*, vol. 7515. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[ap] J. Knodel, R. Koschke, and T. Mende, "Reverse engineering in a reuse context." Kaiserslautern, 2006.

[aq] E. K. Abbasi and P. Heymans, "Reverse engineering web sales configurators," in *IEEE International Conference on Software Maintenance, ICSM*, 2014, pp. 586–589.

[ar] K. Czarnecki, S. She, and A. Wasowski, "Sample spaces and feature models: There and back again," in *Proceedings - 12th International Software Product Line Conference, SPLC 2008*, 2008, pp. 22–31.

[as] M. Acher, B. Baudry, P. Heymans, A. Cleve, and J.-L. Hainaut, "Support for reverse engineering and maintaining feature models," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, 2013, p. 20.

[at] J. Feigenspan, M. Schulze, M. Papendieck, C. Kastner, R. Dachselt, V. Koppen, M. Frisch, and G. Saake, "Supporting program comprehension in large preprocessor-based software product lines," *IET Softw.*, vol. 6, no. 6, p. 488, 2012.

[au] G. Bécan, R. Behjati, A. Gotlieb, and M. Acher, "Synthesis of Attributed Feature Models From Product Descriptions: Foundations," Inria Rennes, Feb. 2015.

[av] T. Ziadi, C. Henard, M. Papadakis, M. Ziane, and Y. Le Traon, "Towards a Language-Independent Approach for Reverse-Engineering of Software Product Lines," in *Sac '14*, 2014, pp. 1064–1071.

[aw] L. Moonen, "Towards evidence-based recommendations to guide the evolution of component-based product families," *Sci. Comput. Program.*, vol. 97, pp. 105–112, Jan. 2013.

[ax] J. Feigenspan and M. Schulze, "Using background colors to support program comprehension in software product lines," in *International Conference on Evaluation and Assessment in Software Engineering*, 2011, pp. 66–75.

[ay] I. Pashov and M. Riebisch, "Using feature modeling for program comprehension and software architecture recovery," *Proc. - 11th IEEE Int. Conf. Work. Eng. Comput. Syst. ECBS 2004*, pp. 406–417, May 2004.

[az] C. Kastner, A. Dreiling, and K. Ostermann, "Variability Mining: Consistent Semi-automatic Detection of Product-Line Features," *IEEE Trans. Softw. Eng.*, vol. 40, no. 1, pp. 67–82, Jan. 2014.

[ba] S. Duszynski, "Visualizing and analyzing software variability with bar diagrams and occurrence matrices," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6287 LNCS, pp. 481–485, Sep. 2010.

[bb] G. Bécan, S. Nasr, M. Acher, and B. Baudry, "WebFML: Synthesizing Feature Models Everywhere," *SPLC-18th International ....* pp. 1–5, 15-Sep-2014.

VI. Références des résultats de la recherche approfondie

1 1ère Itération – références pointant vers des nouvelles publications potentiellement fortement pertinentes

Publication	Liste des références potentiellement pertinantes
[a]	• Loesch, F., and Ploedereder, E. 2007. Restructuring Variability in Software Product Lines using Concept Analysis of Product Configurations. In Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR '07). IEEE Computer Society, Washington, DC, USA, 159-170. • "Ziadi, T., Frias, L., Silva, M., and Ziane, M. 2012. Feature Identification from the Source Code of Product Variants. In Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering (CSMR '12). IEEE Computer Society, Washington, DC, USA, 417-422."
[b]	• D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella, "Automated Refactoring of Object Oriented Code into Aspects," Proc. IEEE 21st Int'l Conf. Software Maintenance, pp. 27-36, 2005. • D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella, "Tool-Supported Refactoring of Existing Object-Oriented Code into Aspects," IEEE Trans. Software Eng., vol. 32, no. 9, pp. 698-717, Sept. 2006. • P. Clements and L.M. Northrop, Software Product Lines: Practices and Patterns. Addison-Wesley, 2002. • K. Czarnecki and K. Pietroszek, "Verifying Feature-Based Model Templates against Well-Formedness OCL Constraints," Proc. Fifth Int'l Conf. Generative Programming and Component Eng., pp. 211-220, 2006. • C. Ka¨stner and S. Apel, "Type-Checking Software Product Lines a Formal Approach," Proc. 23rd Int'l Conf. Automated Software Eng., pp. 258-267, 2008. • J. Liu, D. Batory, and C. Lengauer, "Feature Oriented Refactoring of Legacy Applications," Proc. 28th Int'l Conf. Software Eng., pp. 112-121, 2006. • M. Nassau, S. Oliveira, and M.T. Valente, "Guidelines for Enabling the Extraction of Aspects from Existing Object-Oriented Code," J. Object Technology, vol. 8, no. 3, pp. 1-19, 2009. • M. Nassau and M.T. Valente, "Object-Oriented Transformations for Extracting Aspects," Information and Software Technology, vol. 51, no. 1, pp. 138-149, 2009.
[c]	• LOESCH F., PLOEDEREDER E., "Restructuring Variability in Software Product Lines using Concept Analysis of Product Configurations", KRIKHAAR R. L. VERHOEF C. L. G. A. D., Ed., Proceedings of the 11th European Conference on Software Maintenance and Reengineering, Amsterdam, Netherlands, March 2007, IEEE, p. 159–170. • ZIADI T., FRIAS L., DA SILVA M. A. A., ZIANE M., "Feature Identification from the Source Code of Product Variants", MENS T. CLEVE A. F. R., Ed., Proceedings of the 15th European Conference on Software Maintenance and Reengineering, Los Alamitos, CA, USA, 2012, IEEE, p. 417–422. • CHIKOFSKY E. J., CROSS II J. H., "Reverse Engineering and Design Recovery: A Taxonomy", IEEE Software, vol. 7, num. 1, 1990, p. 13–17, IEEE.
[d]	• Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P., 2012. On extracting feature models from product descriptions. In: Eisenecker, U.W., Apel, S., Gnesi, S. (Eds.), Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25–27, 2012. Proceedings, ACM, pp. 45–54.. • Haslinger, E.N., Lopez-Herrejon, R.E., Egyed, A., 2013. On extracting feature models from sets of valid feature combinations. In: Cortellessa, V., Varró, D. (Eds.), FASE, Vol. 7793 of Lecture Notes in Computer Science. Springer, pp. 53–67
[e]	• K. Kang, J. Lee, P. Donohoe. Feature–Oriented Product Line Engineering. IEEE Software, 19(4):58–65, 2002. • T. von der Maßen, H. Lichter. Modeling Variability by UML Use Case Diagrams. In Proc. of the International Workshop on Requirements Engineering for Product Lines (REPL'02), 2002. • G. Halmans, K. Pohl. Communicating the Variability of a Software-Product Family to Customers. Journal of Software and Systems Modeling 2 (1): 15-36, 2003 • K. Czarnecki, M. Antkiewicz. Mapping Features to models: a template approach based on superimposed variants. In Proc. GPCE'05, LNCS 3676, Springer, pp. 422-437, 2005. • N. Szasz, P. Vilanova. Statecharts and Variabilities. In Proc. of Second International Workshop on Variability Modelling of Software-intensive Systems, Germany, 2008.
[f]	• H. Gomaa. Modeling Software Product Lines with UML. In P. Knauber and G. Succi, editors, SPLW2, pages 27–31, Toronto, Canada, May 2001. IESE. IESEReport. No. 051.01/E • G. Halmans and K. Pohl. Communicating the variability of a software-product family. Software System Model, 3:15–36, 2003. • S. Uchitel and J. Kramer. A workbench for synthesising behaviour models from scenarios. In Proceeding of International Conference on Software Engineering (ICSE 2001), 2001. • S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. IEEE Transaction on Software Engineering, 29(2):99–115, February 2003

	• T. van der Maßen and H. Lichter. Modeling Variability by UML Use Case Diagrams. In Requirement Engineering for Product Lines (REPL02), pages 19–25, September 2002.
[g]	• Acher M., Cleve A., Perrouin G, Heymans P, Vanbeneden C, Collet P, Lahire P. (2012) On extracting feature models from product descriptions. In: VaMoS'12, pp 45–54. ACM • Dietrich C, Tartler R, Schroder-Preikschat W, Lohmann D (2012) A robust approach for variability extraction ¨ from the linux build system. In: SPLC'12, pp 21–30 • Haslinger EN, Lopez-Herrejon RE, Egyed A (2013) On extracting feature models from sets of valid feature combinations. In: FASE'13, LNCS, vol 7793, pp 53–67 • Pleuss A, Botterweck G (2012) Visualization of variability and configuration options. Int J Softw Tools Technol Transfer 14(5):497–510 • Sannier N, Acher M, Baudry B (2013) From Comparison Matrix to Variability Model: The Wikipedia Case Study. In: ASE'13. IEEE • Vacchi E, Combemale B, Cazzola W, Acher M (2014) Automating Variability Model Inference for Component-Based Language Implementations. In: 18th International Software Product Line Conference (SPLC'14)
[h]	• /
[i]	• /
[j]	• D. Batory, D. Benavides, and A. Ruiz-Cortes. Automated analysis of feature models: challenges ahead. Commun. ACM, 49(12):45–47, 2006 • D. S. Batory. Feature models, grammars, and propositional formulas. In 9th Int'l Software Product Line Conference, SPLC 2005, Rennes, France, September 26-29, 2005, volume 3714 of LNCS, pages 7–20. Springer, 2005. • J. Bayer, S. Gerard, Ø. Haugen, J. Mansell, B. MøllerPedersen, J. Oldevik, P. Tessier, J.-P. Thibault, and T. Widen. Consolidated Product Line Variability Modeling, pages 195–241. Springer, 2006. • D. Benavides, P. T. Mart´ın-Arroyo, and A. R. Cortes. Au- ´ tomated reasoning on feature models. In 7th Int'l Conference on Advanced Information Systems Engineering, CAiSE 2005, Porto, Portugal, June 13-17, 2005, volume 3520 of LNCS, pages 491–503. Springer, 2005. • D. Benavides, A. Ruiz-Cortes, P. Trinidad, and S. Segura. ´ A Survey on the Automated Analyses of Feature Models. In Jornadas de Ingenier´ıa del Software y Bases de Datos (JISBD), 2006. • S. Buhne, K. Lauenroth, and K. Pohl. Why is it not suf- ¨ ficient to model requirements variability with feature models. In Int'l Workshop on Automotive Requirements Engineering, AuRE 2006, Nagoya, Japan, September 11, 2004. IEEE Computer Society, 2004. • K. Lee, K. C. Kang, and J. Lee. Concepts and guidelines of feature modeling for product line software engineering. In 7th Int'l Conference on Software Reuse, ICSR-7, Austin, TX, USA, April 15-19, 2002, volume 2319 of LNCS, pages 62–77. Springer, 2002. • M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. Covamof: A framework for modeling variability in software product families. In 3rd Int'l Software Product Line Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, volume 3154 of LNCS, pages 197–213. Springer, 2004 • M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques. Softw. Pract. Exper., 35(8):705–754, July 2005.
[k]	• Czarnecki & M. Antkiewicz (2005). 'Mapping Features to Models: A Template Approach Based on Superimposed Variants'. In Proc. Int'l Conf. Generative Programming and Component Engineering (GPCE), pp. 422–437. Springer. • J. Feigenspan, et al. (2011a). 'FeatureCommander: Colorful #ifdef World'. In Software Product Line Conference, pp. 1–2. ACM Press. paper 3 • J. Feigenspan, et al. (2011b). 'Using Background Colors to Support Program Comprehension in Software Product Lines'. In Proc. Int'l Conf. Evaluation and Assessment in Software Engineering (EASE), pp. 66–75. Institution of Engineering and Technology. • F. Heidenreich, et al. (2008). 'FeatureMapper: Mapping Features to Models'. In Comp. Int'l Conf. Software Engineering (ICSE), pp. 943–944. ACM Press. Demo Paper. • C. K¨astner, et al. (2009b). 'FeatureIDE: Tool Framework for Feature-Oriented Software Development'. In Proc. Int'l Conf. Software Engineering (ICSE), pp. 611–614. IEEE CS. Demo Paper. • M. Krone & G. Snelting (1994). 'On the Inference of Configuration Structures from Source Code'. In Proc. Int'l Conf. Software Engineering (ICSE), pp. 49–57. IEEE CS. • J. Liebig, et al. (2010). 'An Analysis of the Variability in Forty PreprocessorBased Software Product Lines'. In Proc. Int'l Conf. Software Engineering (ICSE), pp. 105–114. ACM Press. • G. Rambally (1986). 'The Influence of Color on Program Readability and Comprehensibility'. In Proc. Technical Symposium on Computer Science Education (SIGCSE), pp. 173–181. ACM Press. • B. Shneiderman & R. Mayer (1979). 'Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results'. International Journal of Parallel Programming 8(3):219–238. • A. von Mayrhauser, et al. (1997). 'Program Understanding Behaviour during Enhancement of Large-scale Software'. Journal of Software Maintenance: Research and Practice 9(5):299–327.
[l]	• N. Andersen. Automatic synthesis of feature models based on satisfiability checking. Master's thesis, IT University of Copenhagen, 2009. • B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: a taxonomy and survey. Journal of Software Maintenance and Evolution: Research and Practice, 2011.

	• T. Kishi and N. Noda. Formal verification and software product lines. Commun. ACM, 49(12), 2006.
[m]	• Giuliano Antoniol and Yann-Gael Gu ¨ eh´ eneuc. Feature identifi- ´ cation: a novel approach and a case study. In Proceedings IEEE International Conference on Software Maintenance (ICSM 2005), pages 357–366, Los Alamitos CA, September 2005. IEEE Computer Society Press. • Kunrong Chen and Vaclav Rajlich. Case study of feature location using ´ dependence graph. In Proceedings IEEE International Conference on Software Maintenance (ICSM), pages 241–249. IEEE Computer Society Press, 2000. • Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Locating features in source code. IEEE Computer, 29(3):210–224, March 2003. • Andrew Eisenberg and Kris De Volder. Dynamic feature traces: Finding features in unfamiliar code. In Proceedings IEEE International Conference on Software Maintenance (ICSM 2004), pages 337–346, Los Alamitos CA, September 2005. IEEE Computer Society Press. • Michael Fischer and Harald Gall. Visualizing feature evolution of largescale software based on problem and modification report data. Journal of Software Maintenance and Evolution: Research and Practice, 16(6):385–403, 2004. • Orla Greevy and Stephane Ducasse. Correlating features and code ´ using a compact two-sided trace analysis approach. In Proceedings of 9th European Conference on Software Maintenance and Reengineering (CSMR'05), pages 314–323, Los Alamitos CA, 2005. IEEE Computer Society. • Orla Greevy, Stephane Ducasse, and Tudor G ´ ˆırba. Analyzing feature traces to incorporate the semantics of change in software evolution analysis. In Proceedings of 21st IEEE International Conference on Software Maintenance (ICSM'05), pages 347–356, Los Alamitos, September 2005. IEEE Computer Society. • Orla Greevy, Michele Lanza, and Christoph Wysseier. Visualizing feature interaction in 3-D. In Proceedings of VISSOFT 2005 (3th IEEE International Workshop on Visualizing Software for Understanding), pages 114–119, September 2005. • A. Hamou-Lhadj, E. Braun, D. Amyot, and T. Lethbridge. Recovering behavioral design models from execution traces. In Proceedings IEEE European Conference on Software Maintenance and Reengineering (CSMR 2005), pages 112–121, Los Alamitos CA, 2005. IEEE Computer Society Press. • Rainer Koschke and Jochen Quante. On dynamic feature location. International Conference on Automated Software Engineering, 2005, pages 86–95, 2005. • [Kuhn et al., 2005b] Adrian Kuhn, Orla Greevy, and Tudor Gˆırba. Applying semantic analysis to feature execution traces. In Proceedings IEEE Workshop on Program Comprehension through Dynamic Analysis (PCODA 2005), pages 48–53, Los Alamitos CA, November 2005. IEEE Computer Society Press. • D. Licata, C.D. Harris, and S. Krishnamurthi. The feature signatures of evolving programs. In Proceedings IEEE International Conference on Automated Software Engineering, pages 281–285, Los Alamitos CA, October 2003. IEEE Computer Society Press. • Adrian Lienhard, Adrian Kuhn, and Orla Greevy. Feature dependency browser – a case-study for rapid prototyping of visualizations for mondrian. In Proceedings IEEE International Workshop on Visualizing Software for Understanding (Vissoft 2007), 2007. • Alok Mehta and George Heineman. Evolving legacy systems features using regression test cases and components. In Proceedings ACM International Workshop on Principles of Software Evolution, pages 190–193, New York NY, 2002. ACM Press.
[n]	• J. Liebig, S. Apel, C. Lengauer, C. Kastner, and M. Schulze, " "An analysis of the variability in forty preprocessor-based software product lines," in 32nd International Conference on Software Engineering (ICSE), 2010.
[o]	• M. Acher, A. Cleve, G. Perrouin, P. Heymans, P. Collet, P. Lahire, and C. Vanbeneden. On Extracting Feature Models From Product Descriptions. In Proceedings of the 6th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2012), pages 45–54. ACM, January 2012. • V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, and C. Lucena. Refactoring product lines. In Proceedings of the 5th International Conference on Generative Programming and Component Engineering (GPCE 2006), pages 201–210. ACM, 2006. • D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated Analysis of Feature Models 20 years Later: a Literature Review. Information Systems, 2010 • F. Loesch and E. Ploedereder. Restructuring variability in software product lines using concept analysis of product configurations. In Proceedings of the 11th European Conference on Software Maintenance and Reengineering, CSMR '07, pages 159–170, Washington, DC, USA, 2007. IEEE Computer Society • N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. In Proceedings of 13th International Software Product Lines Conference (SPLC'09), pages 211–220. ACM, 2009. • T. Ziadi, L. Frias, M. A. Almeida da Silva, and M. Ziane. Feature identification from the source code of product variants. In 16th European Conference on Software Maintenance and Reengineering, CSMR 2012, Szeged, Hungary, March 27-30, 2012, pages 417–422. IEEE, 2012.
[p]	• J. Liebig, S. Apel, C. Lengauer, C. Kästner, and M. Schulze, "An analysis of the variability in forty preprocessor-based software product lines," in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 105-114.
[q]	• Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Derivation of feature component maps by means of concept analysis. In 5th Eu. Conf. On Software Maintenance and Reengineering, pages 176–179, 2001.

	• Felix Loesch and Erhard Ploedereder. Restructuring variability in software product lines using concept analysis of product configurations. In 11th Eu. Conf. on Software Maintenance and Reengineering, pages 159–170, 2007.
[r]	• J. Liebig, S. Apel, C. Lengauer, C. Kästner, and M. Schulze. An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. In Proc. ICSE, pages 105–114. ACM, 2010.
[s]	• /
[t]	• Wilde, N., Buckellew, M., Page, H., Rajlich, V., and Pounds, L., (2003), "A Comparison of Methods for Locating Features in Legacy Software", Journal of Systems and Software, vol. 65, no. 2, February 15, pp. 105 114. • Marcus, A. and Rajlich, V., (2005b), "Panel Summary: Identifications of Concepts, Features, and Concerns in Source Code", in Proceedings of 21st IEEE International Conference on Software Maintenance (ICSM'05), Budapest, Hungary, September 25-30, pp. 718
[u]	• M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. Proceedings of VaMoS '12, pages 45–54, 2012 • D. Benavides, S. Segura, and A. R. Cort´es. Automated analysis of feature models 20 years later: A literature review. Inf. Syst., 35(6):615–636, 2010. • K. Chen, W. Zhang, H. Zhao, and H. Mei. An approach to constructing feature models based on requirements clustering. In Proceedings of RE'05, pages 31–40, 2005. • E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed. On extracting feature models from sets of valid feature combinations. In Proceedings of FASE'13, pages 53–67, 2013.
[v]	• M. Jehle, "Feature unweaving: Semi-automated aspect extraction in product line requirements engineering," Master's thesis, University of Zurich, 2010.
[w]	• V. Alves et al. Refactoring product lines. In GPCE '06, pages 201–210. ACM Press.
[x]	• J. Rubin and M. Chechik. A Survey of Feature Location Techniques. In I. Reinhartz-Berger et al., editor, Domain Engineering: Product Lines, Conceptual Models, and Languages. Springer, To appear. • J. Rubin, A. Kirshin, G. Botterweck, and M. Chechik. Managing Forked Product Variants. In Proc. Of SPLC'12, 2012.
[y]	/
[z]	• K. Czarnecki and A. W , asowski. Feature diagrams and logics: There and back again. In SPLC'07, pages 23–34, 2007. • Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In Proc. of ICSE'11, pages 181–190. ACM, 2011. • I. John. Capturing product line information from legacy user documentation. In Software Product Lines, pages 127–159. Springer, 2006. • N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. In SPLC'09, volume 446 of ICPS, pages 211–220. ACM, 2009.
[aa]	/
[ab]	• J. Feigenspan. Empirical Comparison of FOSD Approaches Regarding Program Comprehension – A Feasibility Study. Master's thesis, University of Magdeburg, 2009. • C. Kastner, S. Apel, and D. Batory. A Case Study Implement- ¨ ing Features Using AspectJ. In Proc. Int'l Software Product Line Conference (SPLC), pages 223–232. IEEE CS, 2007. • M. Mezini and K. Ostermann. Variability Management with Feature-Oriented Programming and Aspects. In FSE '04: Proceedings of the 12th International Symposium on Foundations of Software Engineering, pages 127–136. ACM Press, 2004.
[ac]	/
[ad]	• J. Liebig, S. Apel, C. Lengauer, C. Kästner, and M. Schulze, "An analysis of the variability in forty preprocessor-based software product lines," in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 105- 114.
[ae]	• Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P. and Lahire, P. 2012. On extracting feature models from product descriptions. In Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS '12). ACM, New York, NY, USA, 45-54. • Antoniol, G., Canfora, G., Casazza, G., Lucia, A. and Merlo, E. 2002. Recovering Traceability Links between Code and Documentation. IEEE Trans. Softw. Eng. 28, 10 (October 2002), 970-983. • Ghanam, Y. and Maurer, F. 2010. Linking feature models to code artifacts using executable acceptance tests. In Proceedings of the 14th international conference on Software product lines: going beyond (SPLC'10), Jan Bosch and Jaejoon Lee (Eds.). Springer-Verlag, Berlin, Heidelberg, 211-225 • Ramesh, B. and Jarke, M. 2001. Toward Reference Models for Requirements Traceability. IEEE Trans. Softw. Eng. 27, 1 (January 2001), 58-93. • Ziadi, T., Frias, L., Silva, M. and Ziane, M. 2012. Feature Identification from the Source Code of Product Variants. 16th European Conference on Software Maintenance and Reengineering. 417-422
[af]	• D. Faust, and C. Verhoef, "Software product line migration and deployment," Journal of Software Practice and Experiences, vol. 33(10), pp. 933-955, Aug. 2003, doi: 10.1002/spe.530.

[ag]	•] G.C. Murphy, A. Lai, R.J. Walker, M.P. Robillard, Separating features in source code: an exploratory study, in: Proceedings of the 23rd international Conference on Software Engineering, ICSE, Toronto, Ontario, Canada, May 12–19, IEEE Computer Society, Washington, DC, 2001, pp. 275–284. • J. Liu, D. Batory, C. Lengauer, Feature oriented refactoring of legacy applications, in: Proceedings of the 28th international Conference on Software Engineering, ICSE '06, Shanghai, China, May 20–28, ACM, New York, NY, 2006, pp. 112–121. • A. Mehta, G.T. Heineman, Evolving legacy system features into fine-grained components, in: Proceedings of the 24th international Conference on Software Engineering, ICSE'02, Orlando, Florida, May 19–25, ACM, New York, NY, 2002, pp. 417–427. • O. Greevy, S. Ducasse, Correlating features and code using a compact two-sided trace analysis approach, in: Proceedings of the Ninth European Conference on Software Maintenance and Reengineering, CSMR, March 21–23, IEEE Computer Society, Washington, DC, 2005, pp. 314–323. • A.D. Eisenberg, K. De Volder, Dynamic feature traces: Finding features in unfamiliar code, in: ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance, vol. 0, IEEE Computer Society, Washington, DC, USA, 2005, pp. 337–346. • W.E. Wong, J.R. Horgan, S.S. Gokhale, K.S. Trivedi, Locating program features using execution slices, in: ASSET '99: Proceedings of the 1999 IEEE Symposium on Application—Specific Systems and Software Engineering and Technology, IEEE Computer Society, Washington, DC, USA, 1999, p. 194+. • T. Eisenbarth, R. Koschke, D. Simon, Locating features in source code, IEEE Trans. Softw. Eng. 29 (3) (2003) 210–224. • K. Chen, V. Rajlich, Case study of feature location using dependence graph, in: Proceedings of the 8th international Workshop on Program Comprehension, IWPC, June 10–11, IEEE Computer Society, Washington, DC, 2000, p. 241.
[ah]	• G. Antoniol and Y.-G. Gueheneuc. Feature identification: a novel approach and a case study. In Proceedings of the IEEE International Conference on Software Maintenance (ICSM), 2005. • D. Edwards, S. Simmons, and N. Wilde. An approach to feature location in distributed systems. Journal of Systems and Software, 2006. • T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. IEEE Transactions on Software Engineering, 2003. • A. Eisenberg and K. De Volder. Dynamic feature traces: finding features in unfamiliar code. In Proceedings of the IEEE International Conference on Software Maintenance (ICSM), 2005. • D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. IEEE Transactions on Software Engineering, 2007. • J. Wang, X. Peng, Z. Xing, and W. Zhao. Improving feature location practice with multi-faceted interactive exploration. In Proceedings of the 2013 International Conference on Software Engineering (ICSE), 2013. • N. Wilde and M. C. Scully. Software reconnaissance: mapping program features to code. Journal of Software Maintenance, 1995. • W. E. Wong, J. R. Horgan, S. S. Gokhale, and K. S. Trivedi. Locating program features using execution slices. In Proceedings of the 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology (ASSET), 1999
[ai]	• F. Bachmann and L. Bass. Managing variability in software architectures. SIGSOFT Softw. Eng. Notes, 26:126–132, May 2001. • D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated Analysis of Feature Models 20 years Later: a Literature Review. Information Systems, 2010. • I. John. Capturing product line information from legacy user documentation. In Software Product Lines, pages 127–159. Springer, 2006. • A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, and G. Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In RE'07, pages 243–253, 2007 • N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. In SPLC'09, volume 446 of ICPS, pages 211–220. ACM, 2009.
[aj]	• M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions," in Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems, ser. VaMoS '12. New York, NY, USA: ACM, 2012, pp. 45-54. • S. Deelstra and M. Sinnema, "Managing the complexity of variability in software product families," Ph.D. dissertation, Institute of Mathematics and Computing Science, University of Groningen, Jul. 2008. • A. Lora-Michiels, C. Salinesi, and R. Mazo, "A method based on association rules to construct product line models," in VaMoS'10, 2010, pp. 147-150. • B. Zhang and M. Becker, "Mining complex feature correlations from software product line configurations," in Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, ser. VaMoS '13. New York, NY, USA: ACM, 2013.

[ak]	• D. Benavides, S. Segura, and A. Ruiz-Cort´es. Automated analysis of feature models 20 years later: a literature review. Information Systems, 35(6), 2010. • N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. In SPLC, 2009.
[al]	• N. Weston, R. Chitchyan, and A. Rashid, "A framework for constructing semantically composable feature models from natural language requirements," in SPLC, ser. ACM International Conference Proceeding Series, D. Muthig and J. D. McGregor, Eds., vol. 446. ACM, 2009, pp. 211– 220. • D. Benavides, S. Segura, and A. R. Cortes, "Automated analysis of ´ feature models 20 years later: A literature review," Inf. Syst., vol. 35, no. 6, pp. 615–636, 2010.
[am]	• Al-Msie'deen, R., Seriai, A., Huchard, M., Urtado, C., Vauttier, S., Salman, H.E.: Mining features from the object-oriented source code of a collection of software variants using formal concept analysis and latent semantic indexing. In: SEKE '13. (2013) 244–249 • Al-Msie'deen, R., Seriai, A., Huchard, M., Urtado, C., Vauttier, S.: Documenting the mined feature implementations from the object-oriented source code of a collection of software product variants. In: SEKE '14. (2014) 264–269 • Haslinger, E.N.: Reverse engineering feature models from program configurations. Master's thesis, Johannes Kepler University Linz, Linz, Austria (September 2012) • Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P.: On extracting feature models from product descriptions. In: VaMoS '12, New York, NY, USA, ACM (2012) 45–54
[an]	• Mathieu Acher, Anthony Cleve, Gilles Perrouin, Patrick Heymans, Charles Vanbeneden, Philippe Collet, and Philippe Lahire. On extracting feature models from product descriptions. In VaMoS'12, pages 45 54. ACM, 2012. • Guillaume Bécan. Reverse engineering feature models in the real. Bibliographic report of master research internship, 2013. • Evelyn Nicole Haslinger, Roberto Erick Lopez-Herrejon, and Alexander Egyed. On extracting feature models from sets of valid feature combinations. In Vittorio Cortellessa and Dániel Varró, editors, FASE, volume 7793 of Lecture Notes in Computer Science, pages 53 67. Springer, 2013. • Julia Rubin and Marsha Chechik. Locating distinguishing features using di sets. In the 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012, pages 242 245, New York, NY, USA, 2012. ACM. • N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. In SPLC'09, pages 211 220. ACM, 2009.
[ao]	• Weston, N., Chitchyan, R., Rashid, A.: A framework for constructing semantically composable feature models from natural language requirements. In Muthig, D., McGregor, J.D., eds.: SPLC. Volume 446 of ACM International Conference Proceeding Series., ACM (2009) 211 220 • Acher, M., Cleve, A., Perrouin, G., Heymans, P., Vanbeneden, C., Collet, P., Lahire, P.: On extracting feature models from product descriptions. [25] 45 54
[ap]	/
[aq]	• A. Classen, Q. Boucher, and P. Heymans, "A text-based approach to feature modelling: Syntax and semantics of tvl," Sci. Comput. Program., vol. 76, no. 12, pp. 1130–1143, 2011
[ar]	• D. S. Batory, D. Benavides, and A. R. Cortés. Automated analysis of feature models: challenges ahead. Commun. ACM, 2006. • T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, J. Møller, and H. Hulgaard. Fast backtrack-free product configuration using a precompiled solution space representation. In PETO '04. DTU-tryk, 2004 • S. Robak and A. Pieczynski. Employment of fuzzy logic in ´ feature diagrams to model variability in software families. J. Integr. Des. Process Sci., 2003.
[as]	• D. Benavides, S. Segura, and A. Ruiz-Cortes, "Automated analysis of feature models 20 years later: a literature review," Information Systems, vol. 35, no. 6, 2010. • N. Weston, R. Chitchyan, and A. Rashid, "A framework for constructing semantically composable feature models from natural language requirements," in SPLC'09. ACM, 2009, pp. 211–220. • M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions," in VaMoS'12. ACM, 2012, pp. 45–54. • A. Rabkin and R. Katz, "Static extraction of program configuration options," in ICSE'11. ACM, 2011, pp. 131–140.
[at]	• J. Feigenspan, C. Kastner, S. Apel, and T. Leich. How to Compare Program Comprehension in FOSD Empirically - An Experience Report. ¨ In Proc. Int'l Workshop on Feature-Oriented Software Development (FOSD), pages 55–62. ACM Press, 2009. • J. Feigenspan, M. Schulze, M. Papendieck, C. Kastner, R. Dachselt, V. K ¨ oppen, and M. Frisch. Using Background Colors to Support ¨ Program Comprehension in Software Product Lines. In Proc. Int'l Conf. Evaluation and Assessment in Software Engineering (EASE), pages 66–75. Institution of Engineering and Technology, 2011. • F. Heidenreich, J. Kopcsek, and C. Wende. FeatureMapper: Mapping Features to Models. In Comp. Int'l Conf. Software Engineering (ICSE), pages 943–944. ACM Press, 2008. • R. Tiarks. What Programmers Really Do: An Observational Study. In Workshop Software Reengineering (WSR), pages 36–37, 2011.

[au]	• M. H. ter Beek, S. Gnesi, and F. Mazzanti, "VMC: A tool for the analysis of variability in software product lines," ERCIM News, vol. 2013, no. 93, 2013. [Online]. Available: http://ercim-news.ercim.eu/en93/ri/ vmc-a-tool-for-the-analysis-of-variability-in-software-product-lines • C. Seidl, I. Schaefer, and U. Aßmann, "Capturing variability in space and time with hyper feature models," in VaMoS. ACM, 2014, p. 6. • N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. S. Batory, M. Rosenmüller, and G. Saake, "Predicting performance via automated feature-interaction detection," in ICSE, M. Glinz, G. C. Murphy, and M. Pezzè, Eds. IEEE, 2012, pp. 167–177. • M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions," in VaMoS'12. ACM, 2012, pp. 45–54.
[av]	• M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. In VaMoS, pages 45–54, 2012. • C. Berger, H. Rendel, and B. Rumpe. Measuring the ability to form a product line from existing products. In VaMoS, volume 37, pages 151–154, 2010. • Y. Xue, Z. Xing, and S. Jarzabek. Feature location in a collection of product variants. In WCRE, pages 145–154, 2012. • B. Zhang and M. Becker. Recovar: A solution framework towards reverse engineering variability. In Proc. PLEASE, pages 45–48, 2013. • T. Ziadi, L. Frias, M. A. A. da Silva, and M. Ziane. Feature identification from the source code of product variants. In CSMR, pages 417–422, 2012.
[aw]	/
[ax]	/
[ay]	/
[az]	• D. Benavides, S. Seguraa, and A. Ruiz-Cortes, "Automated Analysis of Feature Models 20 Years Later: A Literature Review," Information Systems, vol. 35, no. 6, pp. 615-636, 2010. • B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A Systematic Survey of Program Comprehension through Dynamic Analysis," IEEE Trans. Software Eng., vol. 35, no. 5, pp. 684-702, Sept./Oct. 2009. • B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature Location in Source Code: A Taxonomy and Survey," J. Software: Evolution and Process, vol. 25, no. 1, pp. 53-95, 2012. • A. Rabkin and R. Katz, "Static Extraction of Program Configuration Options," Proc. Int'l Conf. Software Eng. (ICSE), pp. 131-140, 2011. • E. Reisner, C. Song, K.-K. Ma, J.S. Foster, and A. Porter, "Using Symbolic Evaluation to Understand Behavior in Configurable Software Systems," Proc. Int'l Conf. Software Eng. (ICSE), pp. 445- 454, 2010. • N. Wilde and M.C. Scully, "Software Reconnaissance: Mapping Program Features to Code," J. Software Maintenance: Research and Practice, vol. 7, no. 1, pp. 49-62, 1995.
[ba]	/
[bb]	• M. Acher, P. Collet, P. Lahire, and R. B. France. Familiar: A domain-specific language for large scale management of feature models. Sci. Comput. Program., 78(6):657–681, 2013. • M. Acher, B. Combemale, P. Collet, O. Barais, P. Lahire, and R. B. France. Composing your compositions of variability models. In MoDELS'13, pages 352–369, 2013. • D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated analysis of feature models 20 years later: a literature review. Information Systems, 35(6), 2010. • T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wasowski. A survey of variability modeling in industrial practice. In VaMoS'13. ACM, 2013. • K. Chen, W. Zhang, H. Zhao, and H. Mei. An approach to constructing feature models based on requirements clustering. In RE'05, pages 31–40, 2005. • K. Czarnecki, C. H. P. Kim, and K. T. Kalleberg. Feature models are views on ontologies. In SPLC '06, pages 41–51. IEEE, 2006. • N. Niu and S. M. Easterbrook. Extracting and modeling product line functional requirements. RE, 8:155–164, 2008. • R. Pohl, K. Lauenroth, and K. Pohl. A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In ASE'11, pages 313–322, 2011 • A. Rabkin and R. Katz. Static extraction of program configuration options. In ICSE'11, pages 131–140. ACM, 2011. • N. Sannier, M. Acher, and B. Baudry. From comparison matrix to variability model: The wikipedia case study. In ASE, pages 580–585, 2013. • B. Zhang and M. Becker. Mining complex feature correlations from software product line configurations. In Gnesi et al. [20], page 19.

2 1ère itération – références retenues

[it1.a] N. Weston, R. Chitchyan, and A. Rashid, "A framework for constructing semantically composable feature models from natural language requirements," pp. 211–220, Aug. 2009.

[it1.b] C. Dietrich, R. Tartler, W. Schröder-Preikschat, and D. Lohmann, "A robust approach for variability extraction from the linux build system," in ACM International Conference Proceeding Series, 2012, vol. 1, pp. 21–30.

[it1.c] J. Rubin and M. Chechik, "A Survey of Feature Location Techniques," Domain Engineering, 2013. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-36654-3_2. [Accessed: 27-Apr-2015].

[it1.d] O. Greevy and S. Ducasse, "Correlating features and code using a compact two-sided trace analysis approach," in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2005, pp. 314–323.

[it1.e] T. Ziadi, L. Frias, M. A. A. Da Silva, and M. Ziane, "Feature identification from the source code of product variants," in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2012, pp. 417–422.

[it1.f] J. Feigenspan, M. Papendieck, C. Kästner, M. Frisch, and R. Dachselt, "FeatureCommander: colorful \#ifdef world," Proceedings of the 15th International Software Product Line Conference, Volume 2, 2011. [Online]. Available: http://doi.acm.org/10.1145/2019136.2019192. [Accessed: 27-Apr-2015].

[it1.g] N. Sannier, M. Acher, and B. Baudry, "From comparison matrix to Variability Model: The Wikipedia case study," in 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013 - Proceedings, 2013, pp. 580–585.

[it1.h] K. Czarnecki and M. Antkiewicz, "Mapping features to models: A template approach based on superimposed variants," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2005, vol. 3676 LNCS, pp. 422–437.

[it1.i] B. Zhang and M. Becker, "Mining complex feature correlations from software product line configurations," in Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, 2013, p. 19.

[it1.j] R. Al-Msie'Deen, a. D. Seriai, M. Huchard, C. Urtado, and S. Vauttier, "Mining features from the object-oriented source code of software variants by combining lexical and structural similarity," in Proceedings of the 2013 IEEE 14th International Conference on Information Reuse and Integration, IEEE IRI 2013, 2013, pp. 586–593.

[it1.k] R. Koschke and J. Quante, "On dynamic feature location," in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering SE - ASE '05, 2005, pp. 86–95.

[it1.l] A. Hamou-Lhadj, E. Braun, D. Amyot, and T. Lethbridge, "Recovering behavioral design models from execution traces," in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2005, pp. 112–121.

[it1.m] F. Loesch and E. Ploedereder, "Restructuring variability in software product lines using concept analysis of product configurations," Proc. Eur. Conf. Softw. Maint. Reengineering, CSMR, pp. 159–168, 2007.

[it1.n] E. N. Haslinger, "Reverse Engineering Feature Models from Program Configurations," 2012. [Online]. Available: http://www.jku.at/JKU_Site/JKU/isse/content/e104861/e105613/e179647/e179648/DA_Haslinger.pdf. [Accessed: 04-Apr-2015].

[it1.o] O. Greevy, M. Lanza, and C. Wysseier, "Visualizing feature interaction in 3-D," in Proceedings - VISSOFT 2005: 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2005, pp. 114–119.

3 2ème itération – références retenues

[it2.a] R. AL-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, *Feature Location in a Collection of Software Product Variants Using Formal Concept Analysis*, vol. 7925. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[it2.b] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 234–243.

[it2.c] S. Deelstra and M. Sinnema, "Managing the complexity of variability in software product families," 2008. [Online]. Available: http://dissertations.ub.rug.nl/faculties/science/2008/m.sinnema/?pLanguage=en&pFullItemRecord=ON. [Accessed: 27-Apr-2015].

VII. Exemples d'exécutions FAMILIAR

Voici 3 exemples fournis par le projet FAMILIAR et interprétés sur une machine Windows 8.1 64bits Java 1.8.4 exclusivement pour ce mémoire:

1 Validation de feature models

1.1 Code source

```
1 fm1 = FM (A : [B] [C] ; B -> !C ; B and C ; )
2 b1 = isValid fm1
```

1.2 Interprétation

```
1 fm1> fm1 = FM (A : [B] [C] ; B -> !C ; B and C ; )
2 fm1: (FEATURE_MODEL) A: [C] [B] ;
3 (B -> !C);
4 (B & C);
5 fm1> b1 = isValid fm1
6 b1: (BOOLEAN) false
7 fm1>
```

1.3 Constatation

Le script propose la création d'un feature model ayant 2 contraintes contradictoires. (L.1)
Le programme mentionne en effet un échec à la validation (L.6).

2 Comparaison de feature models

2.1 Code source

```
1 fm1 = FM(A : B [C];)
2 fm2 = FM(A : B; B : [C];)
3 compare fm1 fm2
```

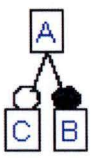
2.2 Interprétation

```
1 fm1> fm2 = FM(A : B; B : [C];)
2 fm2: (FEATURE_MODEL) A: B ;
3 B: [C] ;
4 fm1> fm1 = FM(A : B [C];)
5 fm1: (FEATURE_MODEL) A: [C] B ; fm1> compare fm1 fm2
6 res3: (STRING) REFACTORING
```

2.3 Constatation

Le 1er feature model déclaré (L.1):

Le second feature model déclaré (L.4) :



La fonction « compare » permet de comparer 2 feature models. Le programme retourne « REFACTORING » (L.6) signifiant l'égalité entre les 2 feature models.

3 Fusion de feature models

3.1 Code source

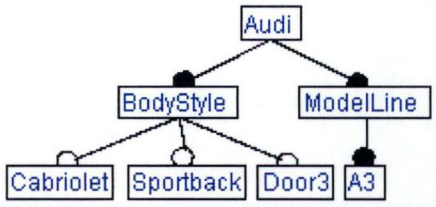
```
1 fm1 = FM (Audi: ModelLine BodyStyle ; ModelLine : A3 ; BodyStyle : [Door3] [Sportback] [Cabriolet] ; )
2 fm2 = FM (Audi: ModelLine BodyStyle ; ModelLine : A4 ; BodyStyle : [Saloon] [Avant] [AllroadQuattro] ; )
3 fm3 = FM (Audi: ModelLine BodyStyle ; ModelLine : A1 ; BodyStyle : [Door3] [Sportback] ; )
4
5 fmAudiS = merge sunion fm*
```

3.2 Interprétation

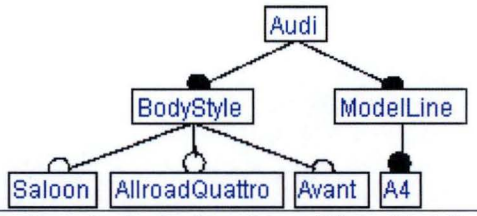
```
1  fml> fm1 = FM (Audi: ModelLine BodyStyle ; ModelLine : A3 ; BodyStyle : [Door3] [Sportback] [Cabriolet] ; )
2  fm1: (FEATURE_MODEL) Audi: BodyStyle ModelLine ;
3  BodyStyle: [Cabriolet] [Sportback] ["Door3"] ;
4  ModelLine: "A3" ;
5  fml> fm2 = FM (Audi: ModelLine BodyStyle ; ModelLine : A4 ; BodyStyle : [Saloon] [Avant] [AllroadQuattro] ; )
6  fm2: (FEATURE_MODEL) Audi: BodyStyle ModelLine ;
7  BodyStyle: [Saloon] [AllroadQuattro] [Avant] ;
8  ModelLine: "A4" ;
9  fml> fm3 = FM (Audi: ModelLine BodyStyle ; ModelLine : A1 ; BodyStyle : [Door3] [Sportback] ; )
10 fm3: (FEATURE_MODEL) Audi: BodyStyle ModelLine ;
11 BodyStyle: [Sportback] ["Door3"] ;
12 ModelLine: "A1" ;
13
14 fml> fmAudiS = merge sunion fm*
15 fmAudiS: (FEATURE_MODEL) Audi: BodyStyle ModelLine ;
16 BodyStyle: (AllroadQuattro|Cabriolet)? (Saloon|Sportback)? (Avant|"Door3")? ;
17 ModelLine: ("A4"|"A3"|"A1") ;
18 ("A4" -> !"Door3");
19 (AllroadQuattro -> "A4");
20 (AllroadQuattro -> !Sportback);
21 (Saloon -> "A4");
22 (Cabriolet -> "A3");
23 (Avant -> "A4");
24 ("A4" -> !Sportback);
25 (Avant -> !Sportback);
```

3.3 Constatation

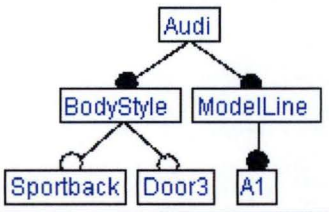
FM Audi A3 (L.1):



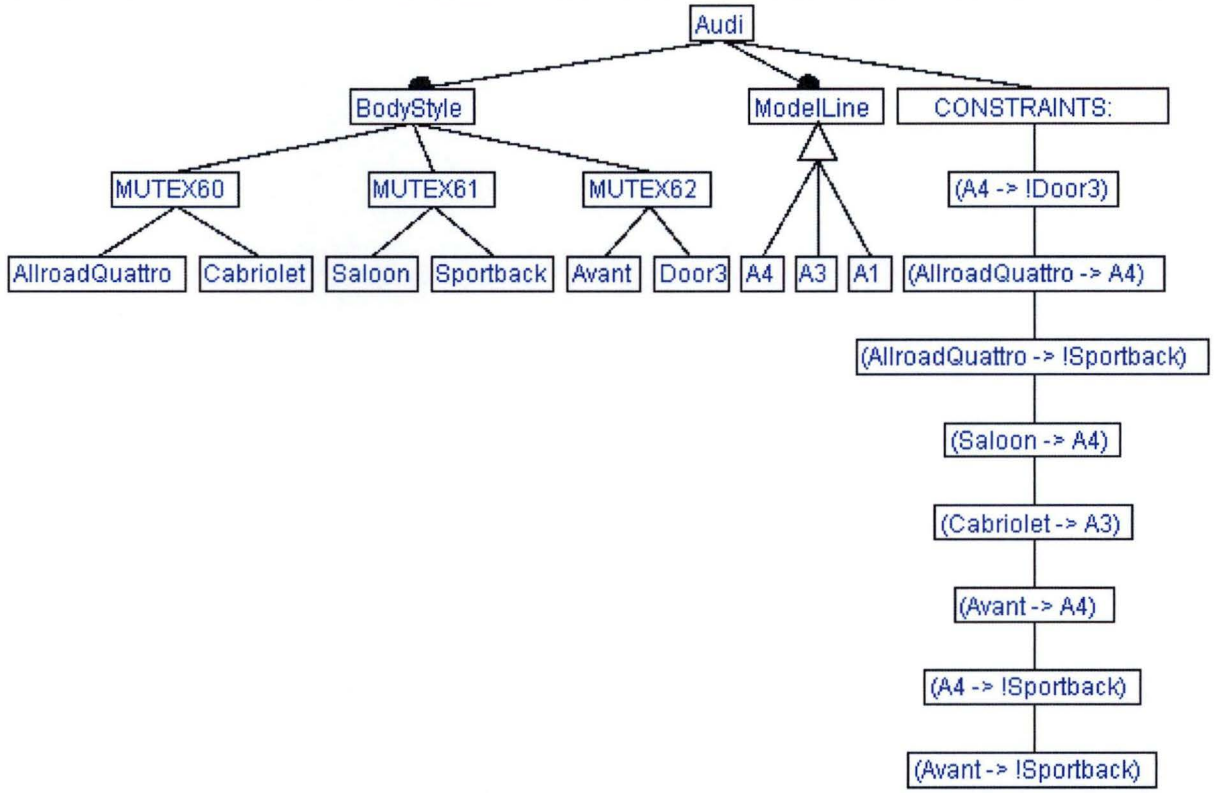
FM Audi A4 (L.5):



FM Audi A1 (L.9):



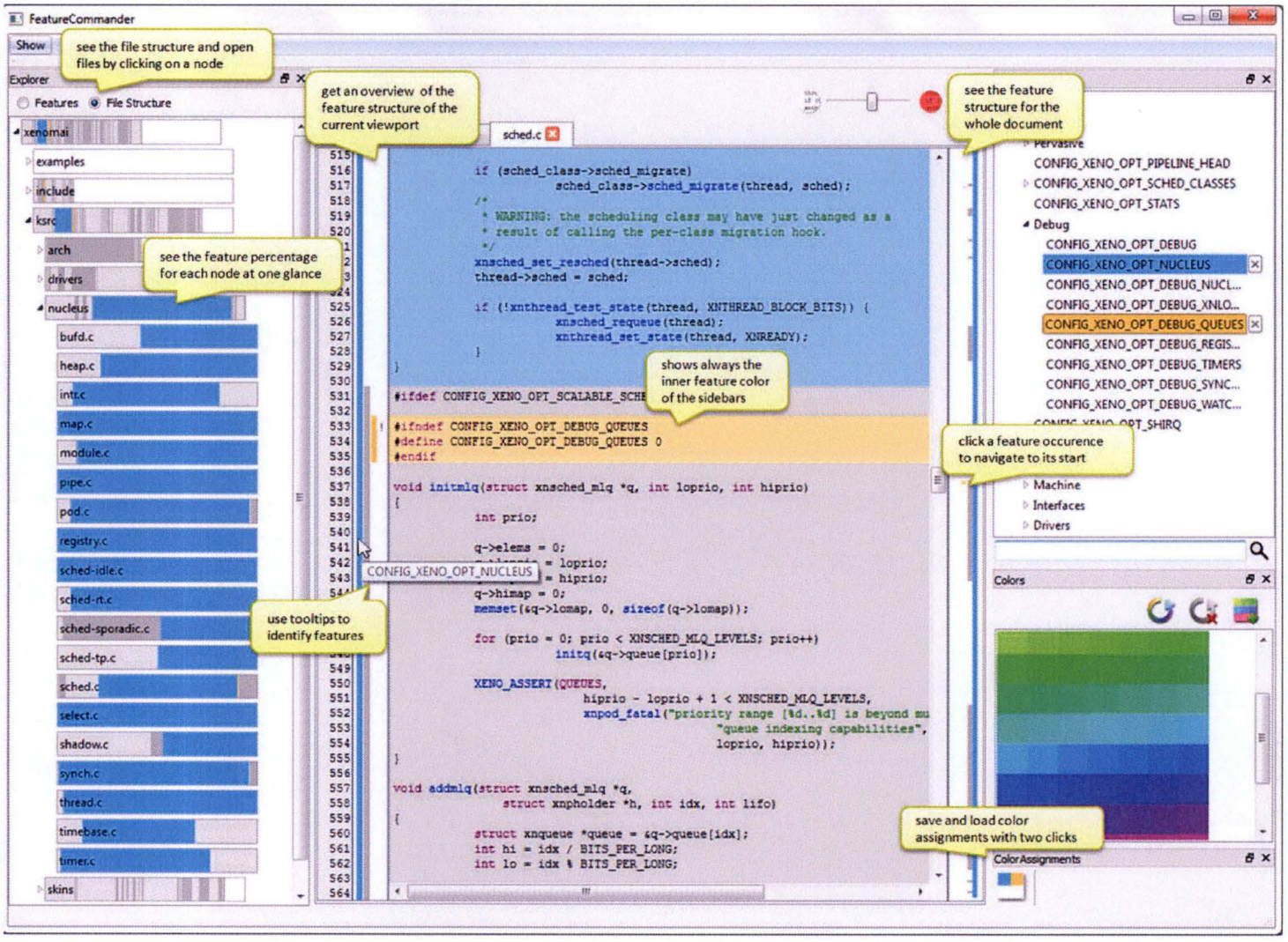
Résultat de la fusion des 3 FMs (L.14):

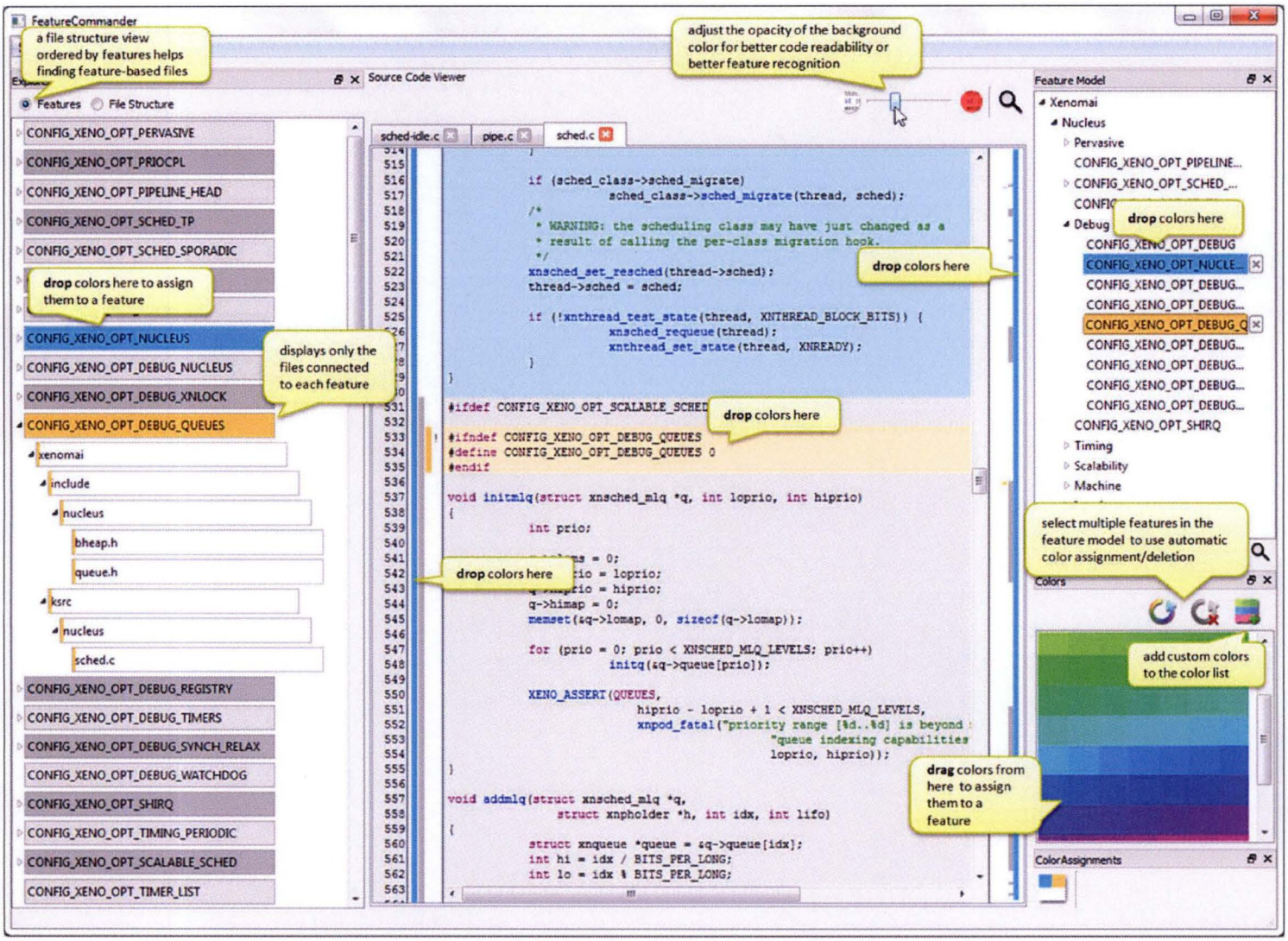


Il est possible de constater une fusion complète avec prise en compte des multiples contraintes. Ce processus se base sur le nom des features.

VIII. Feature Commander – screenshots

Les 2 screenShots commentés suivants proviennent directement du site éditeur (http://www.infosun.fim.uni-passau.de/spl/janet/xenomai)





IX. Events sur lesquels se sont basés Marciuska et Al.

Table 1.1. Mouse Events

Event	Description
onclick	Event occurs when the user clicks on an element
ondblclick	Event occurs when the user double-clicks on an element
onmousedown	Event occurs when a user presses a mouse button over an element
onmousemove	Event occurs when the pointer is moving while it is over an element
onmouseover	Event occurs when the pointer is moved onto an element
onmouseout	Event occurs when a user moves the mouse pointer out of an element
onmouseup	Event occurs when a user releases a mouse button over an element

Table 1.2. Keyboard Events

Event	Description
onkeydown	Event occurs when the user is pressing a key
onkeypress	Event occurs when the user presses a key
onkeyup	Event occurs when the user releases a key

Table 1.3. Frame and Object Events

Event	Description
onabort	Event occurs when an image is stopped from loading before completely loaded (for object)
onerror	Event occurs when an image does not load properly
onload	Event occurs when a document, frameset, or object has been loaded
onresize	Event occurs when a document view is resized
onscroll	Event occurs when a document view is scrolled
onunload	Event occurs once a page has unloaded (for body and frameset)

Table 1.4. Form Events

Event	Description
onblur	Event occurs when a form element loses focus
onchange	Event occurs when the content of a form element, the selection, or the checked state have changed (for input, select, and textarea)
onfocus	Event occurs when an element gets focus (for label, input, select, textarea, and button)
onreset	Event occurs when a form is reset
onselect	Event occurs when a user selects some text (for input and textarea)
onsubmit	Event occurs when a form is submitted